US
University of Sussex

Evolutionary Approaches to Optimisation in Rough Machining

Alexander Wainwright Churchill

Submitted for the degree of Doctor of Philosophy

University of Sussex

September 2013

Declaration

I hereby declare that this thesis has not been and will not be, submitted in whole or in part to another University for the award of any other degree.

Signature: ……………………………………………

Alexander Wainwright Churchill

Acknowledgements

I would like to thank the many people who have helped me throughout my PhD studies. To my parents and family, thank you for your love and support, and invaluable help with proof reading. To Adrian Thompson, thank you for offering me this amazing opportunity. To my supervisors, Andy Philippides and Phil Husbands, thank you for your support, knowledge, guidance and general kindness. To Gerry O'Driscoll from Vero Software, thank you so much for creating, funding and spearheading this collaboration, and for looking after me on my trips to Cheltenham. To John Cockerill and all at Vero Software, thank you so much for your help on this project and being so hospitable. Special thanks are due to Steve Youngs from Vero Software, who among other things, developed an interface to Machining Strategist for this project and provided me with invaluable support in running simulations as well as instructing me in CAD/CAM and machining. To Inman Harvey, Luc Berthouze and Chris Thornton at the University of Sussex, thank you for your knowledge and guidance. To Chrisantha Fernando, thank you for your support and generosity. To the Engineering and Physical Sciences Research Council, thank you for funding this research. To my examiners, thank you very much for your helpful and insightful suggestions. To my bandmates, Aschwin, Freddy and Tom, thank you for putting up with my lack of time. To my housemates, Onur, Sam and Shady, thank you for keeping me sane. And finally, to my girlfriend, Maddie Rix, thank you for your love, support and delicious cooking, and for not getting too annoyed with me for working all the time.

Publications

Parts of this thesis have appeared elsewhere in peer-reviewed publications:

Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2012. Metaheuristic approaches to tool selection optimisation. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO '12)*, Terence Soule (Ed.). ACM, New York, NY, USA, 1079-1086.

Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2013. Multi-objectivization of the Tool Selection Problem on a Budget of Evaluations. In *Proceedings of the 7th International Conference on Evolutionary Multi-criterion Optimization (EMO 2013)*, R.C. Purshouse et al. (Ed.). Lecture Notes in Computer Science, Vol. 7811, 600-614, Springer-Verlag, Berlin, Heidelberg.

Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2013. Multi-objective tool sequence and parameter optimization for rough milling applications. In *2013 IEEE Congress on Evolutionary Computation (CEC),* 1475-1482.

Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2013. Tool sequence optimization using synchronous and asynchronous parallel multi-objective evolutionary algorithms with heterogeneous evaluations. In *2013 IEEE Congress on Evolutionary Computation (CEC),* 2924-2931.

Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2013. Tool sequence optimisation using preferential multi-objective search. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion (GECCO '13 Companion)*, Christian Blum (Ed.). ACM, New York, NY, USA, 181-182.

UNIVERSITY OF SUSSEX

Alexander Wainwright Churchill, Doctor of Philosophy

Evolutionary Approaches to Optimisation in Rough Machining

This thesis concerns the use of Evolutionary Computation to optimise the sequence and selection of tools and machining parameters in rough milling applications. These processes are not automated in current Computer-Aided Manufacturing (CAM) software and this work, undertaken in collaboration with an industrial partner, aims to address this. Related research has mainly approached tool sequence optimisation using only a single tool type, and machining parameter optimisation of a single-tool sequence. In a real world industrial setting, tools with different geometrical profiles are commonly used in combination on rough machining tasks in order to produce components with complex sculptured surfaces. This work introduces a new representation scheme and search operators to support the use of the three most commonly used tool types: end mill, ball nose and toroidal. Using these operators, single-objective metaheuristic algorithms are shown to find near-optimal solutions, while surveying only a small number of tool sequences. For the first time, a multi-objective approach is taken to tool sequence optimisation. The process of 'multi-objectivisation' is shown to offer two benefits: escaping local optima on deceptive multimodal search spaces and providing a selection of tool sequence alternatives to a machinist. The multi-objective approach is also used to produce a varied set of near-Pareto optimal solutions, offering different trade-offs between total machining time and total tooling costs, simultaneously optimising tool sequences and the cutting speeds of individual tools. A challenge for using computationally expensive CAM software, important for real world machining, is the time cost of evaluations. An asynchronous parallel evolutionary optimisation system is presented that can provide a significant speed up, even in the presence of heterogeneous evaluation times produced by variable length tool sequences. This system uses a distributed network of processors that could be easily and inexpensively implemented on existing commercial hardware, and accessible to even small workshops.

Table of Contents

Chapter 1

Introduction

1.1 Genesis

Manufacturing is one of the most important contributions to a nation's economy and is the driving force behind technological innovation. In essence it involves the creation of goods for use or sale. The earliest usage of the word denoted an article made by hand (OED, 2013). Since the Industrial Revolution, the connotation of 'made by hand' has largely been superseded by 'made by powerful machines', beginning with the steam powered Boring machine in the 1770s and coming to include the largely computer-automated production lines of the 21st century.

Manufacturing is often described as a process, transforming raw material into a final product in a series of operations. At each stage in the manufacturing cycle, value is added to the produce. For example iron can be melted and processed into steel, which can be machined into an automobile part, adding value at every stage. Even this simple example suggests that there are many different types of process at play, which are combined to create the products that we see every day. These segments of the manufacturing chain can be broadly divided into processing and assembly operations. Processing involves the transformation of a material into a new shape or form, while assembly encompasses the joining together of different components to create a new unit.

Optimisation is of huge concern to the manufacturing industry. The reasons for this are easy to comprehend. By creating products more efficiently, production costs and manufacturing times can be lowered. For consumers this means that the costs of products are reduced benefitting their lives, and the manufacturer is able to produce a greater number of goods at lower cost, increasing productivity. Efficiency gains are sought at every stage in the chain, and optimisation is performed at both a macro and micro level.

This thesis concerns the optimisation of a small but important area of manufacturing called *machining*, which involves the removal of material from a solid. According to the 2012 edition of DeGarmo's *Materials and Processes in Manufacturing*,

> "US industries annually spend well over $100 billion on metal removal operations because the vast majority of manufactured products require machining at some stage in their production." (Black and Kohser, 2012: 533)

The majority of the work in modern day machining is executed on computer-controlled machines. Even in this one process, optimisation is performed at a macro and micro level. In the latter case, the individual paths that a tool takes can be made more efficient, while in the former, the schedule determining which parts should be loaded onto which machines can be modified to ensure the maximum production rate in a workshop. The main focus of this thesis is on a problem that encompasses both these concerns. Referred to as the *Tool Selection Problem* or *Tool Sequence Optimisation*, the goal is to find a sequence of tools that can machine a component efficiently.

Traditionally in machining, parameter selection and tool sequence design have been performed by a human expert using a combination of handbook values and personal experience. However, this is an area for which the resources of computing provide great promise of bringing about notable gains in efficiency, which has come to the attention of forward-looking companies producing software for computer controlled machining. The work in this thesis has been undertaken in collaboration with a Computer Aided Manufacturing (CAM) company, Vero Software. The goal is to explore ways of automating the selection of tools in *rough machining*, assisting a human process planner to locate near optimal solutions.

The lack of automation in CAM software is eloquently described by Vosniakos and Krimpenis,

> "In roughing, in particular, where tools can be changed at will in order to accelerate the cutting process, much improvement could be achieved by the choice of the correct combination of cutting strategy and tools. This is a planning task and, apart from experience (which in the strict sense of optimality is of questionable value), few formal methods could prove helpful in tackling it successfully." (2002: 1)

In the following chapters in this thesis, this optimisation problem is tackled using the same methods as Vosniakos and Krimpenis, *Evolutionary Computation*, an umbrella term for algorithms that take inspiration from the principles of *Natural Selection*, and has been applied to a diverse range of problem domains. The most famous technique from Evolutionary Computation is the *Genetic Algorithm* (Holland, 1975), which is a general-purpose stochastic optimiser. Tool Sequence Optimisation (TSO) is a combinatorial problem, with solutions of variable length, set in a discrete search space. Machining Parameter Optimisation (MPO) operates in a continuous domain. To combine both into one algorithm creates a mixed search space with both discrete and continuous properties. Genetic Algorithms (GAs) are general-purpose stochastic optimisers, which can work in continuous, discrete and mixed domains (Watanabe and Hashem, 2004; Giagkiozis et al., 2013).

Many researchers have used Evolutionary Computation in TSO and MPO problems, for example (Krimpenis and Vosniakos, 2009; Datta and Deb, 2009; Ahmad et al., 2010; Chen and Fu, 2011). Apart from its application to numerous problem domains, other advantages of Evolutionary Computation include its ability to escape local optima, handle linear and non-linear constraints, and its easy extensibility to *multi-objective optimisation* problems (Giagkiozis et al., 2013).

Reviewing the literature in machining, we notice that there are three main areas that have been under-represented. Firstly, there have been very few papers, e.g. (Wang et al., 2005a, Spanoudakis et al., 2008), that have looked at the selection of tool types other than flat end mills. No researchers have included the flexible and industrially prevalent toroidal tool type in TSO systems. Secondly, there has been no multi-objective approach taken to TSO, even though several researchers have looked into multi-objective MPO (e.g. Sardiñas et al., 2006; Datta and Deb, 2009), and the same objective functions apply. Thirdly, there has been very little work in combining MPO with TSO, and no multi-objective work.

The work in this thesis contributes to these three areas. Firstly, with the use of sophisticated CAM software provided by our industrial partner, we are able to create a system that can optimise tool sequences containing the three main types of cutter geometry – flat end mill, toroidal and ball nose (Smid, 2003). This is achieved using a GA and other metaheuristic techniques. The extensibility of Genetic Algorithms means that it is possible to implement a multi-objective optimisation system without modifying the representation scheme or search operators. In the second half of this thesis, we introduce the first multi-objective TSO systems. The first of these provides a selection of tool sequences offering a trade-off between total machining time and surface tolerance. However, it is taken from a *multi-objectivisation* perspective, which is the application of multi-objective techniques to single objective problems (Knowles et al., 2001). This puts a particular emphasis on locating a single solution, utilising multi-objective search to escape local optima. The second system provides a true multi-objective approach, returning a set of solutions offering trade-offs between total machining time and total tooling costs. In this last system, the third underrepresented area in the literature identified above is addressed. A separate, continuous real-valued machining parameter is simultaneously optimised for each tool in the sequence, enabling a wide range of trade-off solutions to be found.

Another advantage of population-based Evolutionary Algorithms is that they are naturally parallelisable (Cantú-Paz, 2000; Giagkiozis et al., 2013). The goal of the work in this thesis is for it to be integrated into CAM software and assist human process planners. In order to handle the computational expense of using sophisticated CAM software, a parallel architecture is employed that could easily be implemented on existing office networks or inexpensive commercially available grid (cloud) services. This is shown to provide good solutions in short amounts of time.

1.2 Thesis Organisation

After the present chapter, this thesis is organised in the following way. In Chapter 2, we present relevant general background information to introduce an unfamiliar reader to machining processes and Evolutionary Computation. We also review related previous research into TSO and MPO before outlining how we will address areas that have been underrepresented or are missing from related work.

In Chapter 3, we present a formal definition of the Tool Selection Problem and describe the methods used to evaluate tool sequence solutions. We also present a system we developed to distribute tool sequence evaluations across multiple machines, which allows for large number of tool sequences to be assessed rapidly and is thus useful for parallelising search, caching search spaces for analysis and testing CAM software in an industrial setting.

In Chapter 4, the search space of a constrained Tool Selection Problem is explored to evaluate the validity of heuristics used in previous research, when applied to tool libraries containing several different tool types. We discover that the use of a number of these heuristics could prevent optimal solutions from being found.

In Chapter 5, four metaheuristic algorithms are applied to optimise tool sequences on multiple parts, using, for the first time, a library containing end mill, toroidal and ball nose tools. These algorithms are a Genetic Algorithm, Stochastic Hill Climber, Random Restart Stochastic Hill Climber and a Hybrid Genetic Algorithm. A new representation scheme and problem specific operators are introduced that support the use of multiple tool types. These algorithms are found to perform extremely well on three of the four parts tested, showing their suitability for solving tool selection problems. Analysis of the search space of the difficult part shows that there are a number of local optima and sharp jumps in fitness, created partly through simulation noise. This provides an interesting test search space that is explored further in Chapter 7.

In Chapter 6, the reader is introduced to Evolutionary Multi-objective Optimisation concepts, multi-objectivisation and preferential multi-objective search. Multi-objective techniques offer two advantages to work in machining - they have been shown to provide a way to escape local optima by following multiple search gradients and they can deliver multiple solutions, offering a trade-off between different objective functions. Comparing NSGA-II (Deb et al., 2002) to a single-objective Genetic Algorithm, multi-objectivisation is shown to improve search performance on the 0/1 Knapsack problem. A novel adaptation to NSGA-II, *Guided Elitism*, is shown to additionally improve performance.

In Chapter 7, the Tool Selection Problem presented in Chapter 5 is multi-objectivised. The multi-objective algorithms are shown to return a selection of Pareto optimal solutions, offering a trade-off between total machining time and total tooling costs. Single and multi-objective algorithms are assessed on the difficult test search space presented in Chapter 5, containing several local optima. Using multi-objective techniques on a fixed budget of evaluations, especially with preferential search through R-NSGA-II (Deb and Sundar, 2006) and the novel Guided Elitism method, is shown to drastically reduce the convergence on sub-optimal solutions, when compared to the single-objective algorithms introduced in Chapter 5.

In Chapter 8, a full multi-objective approach is taken to optimise both tool sequences and cutting speeds, considering total machining time and total tooling costs as separate objectives. This presents the first multi-objective system to simultaneously optimise tool sequences and machining parameters. Five variants of NSGA-II are evaluated on three parts and shown to find solutions that reach a desired minimum surface tolerance with good trade-offs between the two objectives. A novel constraint handling method, *Precedential Objective Ordering*. is shown to regularly find the best solutions.

In Chapter 9, synchronous and asynchronous parallel multi-objective Evolutionary Algorithms are applied to the multi-objectivised Tool Selection Problem tackled previously in Chapter 7, using an enlarged search space. The algorithms are assessed on their ability to find a diverse selection of solutions when using computationally expensive CAM software to evaluate tool sequences in real time. The algorithms, especially the asynchronous version, are shown to provide good results in a time frame that would allow the system to be used in a real world setting. Finally, in the last chapter we summarise the results of the research undertaken for this thesis and outline further directions that could be explored.

1.3 Contributions

We now provide a summary of the main contributions of this thesis:

- We present a distributed processing system, that caches previously processed tool sequence fragments and allows for the parallel evaluation of many tool sequences.
- We demonstrate that evolutionary methods can work on Tool Sequence Optimisation, when using multiple tool types, and present a system that supports, for the first time, tool libraries containing flat endmill, toroidal and ball nose tools. We provide a representation scheme that supports different tool types, variable length sequences and flexible ordering. A novel problem specific mutation operator is also presented which is shown to provide more efficient search.
- We present the first multi-objectivisation approach to Tool Sequence Optimisation, which under a budget of evaluations provides better performance than single-objective algorithms in the presence of several local optima. In this system we also show that preferential search works well on this multi-objectivisation problem. We introduce a novel adaptation to NSGA-II for preferential search, which is also tested on the multi-objectivised 0/1 Knapsack Problem and shown to be more successful than NSGA-II. We also show that in certain situations NSGA-II can perform badly with very small population sizes, and present a modification to NSGA-II's Crowding Distance Assignment to overcome this.
- We present, for the first time, a multi-objective Tool Sequence Optimisation algorithm. This returns solutions that satisfy a surface tolerance constraint and offer trade-offs between two objectives, total machining time and total tooling costs. This is one of very few systems that simultaneously optimise tool sequences and machining parameters, and the first to use a multi-objective approach. We also introduce a novel constraint handling technique that is shown to provide the best performance.
- We present a parallel system that divides computationally expensive Computer Aided Manufacturing software evaluations among many processors in an asynchronous manner. This is shown to perform much faster than a synchronous system, with broadly the same effectiveness in the presence of solution-based time heterogeneity. This means that the techniques presented here can be a practical addition to commercial CAM software, enabling optimisation to be automated in real world industrial situations.

Chapter 2

Background

In this chapter we introduce the main concepts behind machining and Evolutionary Computation. We then introduce Tool Sequence Optimisation and Machining Parameter Optimisation, the problems to which the experimental work in this thesis will be applying Evolutionary Computation. In the final sections, we review related research from the literature and identify under-represented areas before outlining how these will be addressed in the later chapters of this thesis.

2.1 Machining

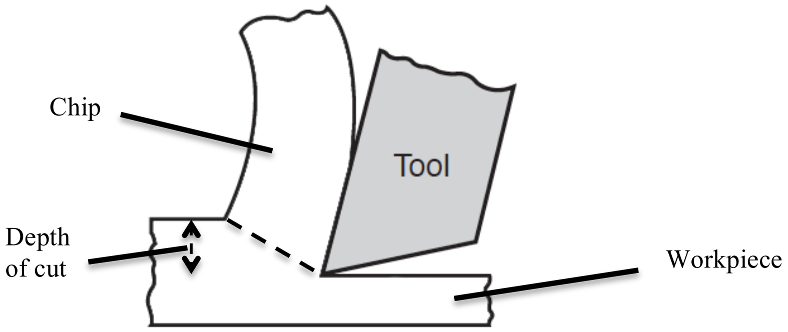
Machining is part of the processing operation known as *material removal* (Groover, 2007). This is itself part of the *shaping* umbrella of processes, where the shape of an initial workpiece gets transformed by the end of the process. In material removal, the starting material is a solid and material is removed to meet a desired final geometry. This is typically achieved by using a machine tool to cut material out of the workpiece, although other techniques such as lasers or abrasive chemicals have also been employed. Conversely, another shaping process is *solidification*. In this case the initial material is in a liquid form and is casted (for metals) or moulded (for plastics) to achieve the final solidified geometry when cooled.

Machining encompasses the group of material removal processes where a sharp cutting tool is used to cut material away from a solid in the form of chips. Generally, a cutting tool removes a chip from the exterior of the workpiece, revealing a new surface underneath. Machining can be used to create a huge variety of shapes, ranging from regular forms such as cylinders and flat planes to extremely intricate complex geometries. Groover writes that,

> "Machining is the most versatile and accurate of all manufacturing processes in its ability to produce a diversity of part geometries and geometric features." (2007: 505)

The three main machining processes are *drilling*, *turning* and *milling*, which are described in more detail below. All three processes involve a machine that operates at high speed. A cutting tool used in machining has one or more sharp edges. In order to effectively remove material, the tool must be made of a material harder than the workpiece. In turning the workpiece rotates while the cutting tool moves laterally. In both drilling and milling the cutting tool rotates. In milling the workpiece moves in a direction perpendicular to the cutting tool, while in drilling the part moves in the same direction as the cutter. The speed of the main rotation is known as the *cutting speed*, and defines the rotation of the part in turning and the cutter in milling and drilling. The *feed* is a secondary motion, which is the movement of the tool in turning and the workpiece in milling and drilling. A third condition, the *depth of cut*,

Figure 2.1. An illustration of material being removed from a workpiece by a tool through sheer deformation. Adapated from (Groover, 2007).



determines how far the cutter penetrates the surface of workpiece with each cut (Kalpakjian and Schmid, 2008).

The force of the hard tool against the material of the softer work surface creates a chip by what is known as *shear deformation*. This occurs where the sharp cutting edge meets the material. The depth of cut determines how far the tool penetrates the original surface, and the initial thickness of the chip. After cutting, the thickness of the chip always expands (Groover, 2007). An illustration of shear deformation is seen in Figure 2.1.

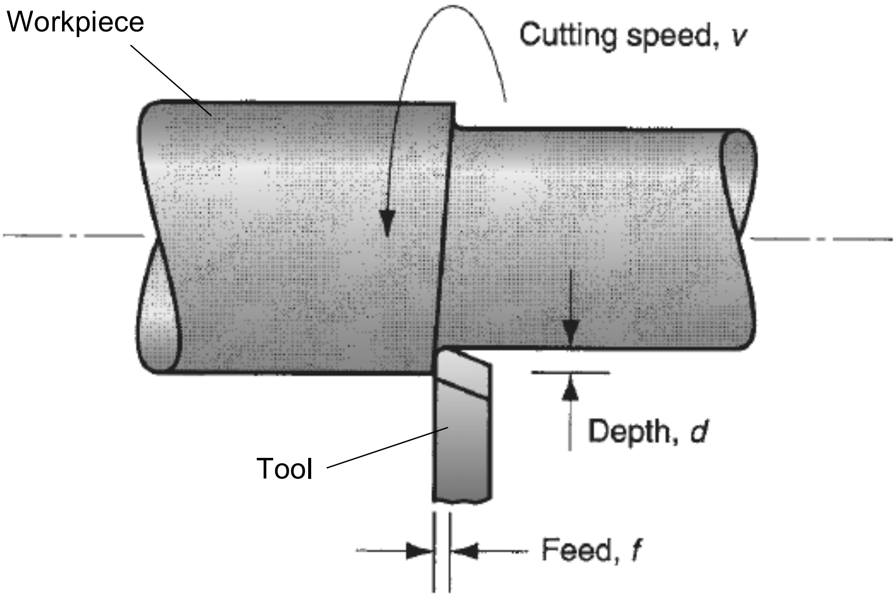
Machined parts are categorised as being either rotational or prismatic. For a rotational part, the work material is rotated against a cutting tool that removes material. To create a prismatic part, the work material moves in a linear motion against a cutting tool that removes material, and may be rotating but does not have to. Giving more primitive examples, a rotational part can be thought of as a clay pot, spinning on a potter's wheel, being shaped by hand. A marble sculpture is an ancient example of a prismatic part, created by hammer and chisel or similar tools.

2.1.1 Turning

In the turning machining process, a workpiece is rotated against a non-rotating cutting tool. The machine that performs this process is known as a lathe, and is mainly used to create curved parts such as shafts, spindles and pins (Kalpakjian and Schmid, 2008). A tool, with one cutting edge, moves in a linear motion, which allows for many different shaping operations. The most common is straight turning. Here, the cutting tool is fed in a straight line against the body of the workpiece, creating a cylinder. In contour turning, the path of the cutting tool can be in a curved line, allowing for contoured shapes. Another operation is facing. Here, the tool is fed against the face of a part, allowing a flat surface to be formed at the end.

There are three important cutting conditions in turning: the cutting speed (v), the feed (f) and the depth of cut (d). The cutting speed is the speed at which the part is being rotated. The feed is the distance that the cutting tool moves against the part at every revolution, and the depth of cut is the distance that the cutting tool penetrates under the surface of the part at any one point. An example of a turning operation is seen in Figure 2.2.

Figure 2.2. Illustrating a turning operation. Showing the depth of cut, *d*, the distance that the tool penetrates the workpiece; the feed, *f*, the distance that the tool moves laterally after every revolution; and the cutting speed, *v*, the speed that the workpiece rotates. Taken from (Groover, 2007) and adapted.



The rotational speed of the part, N, in revolutions per minute, is given by,

$$N = \frac{v}{\pi D} \qquad (2.1)$$

where v is the cutting speed in mm/min and D is the original (pre-machining) diameter of the part in mm. The feed rate (f_r) in mm/min determines the speed that the tool moves linearly along the part. It is a function of the feed and the rotational speed,

$$f_r = Nf \qquad (2.2)$$

where the feed f is in units of mm. The total time taken for a turning operation in minutes, t_m, is often given as,

$$t_m = \frac{L}{f_r} \qquad (2.3)$$

where L is the total length of the cut in mm. Examples of this can be found in (Ostwald and Munoz, 1997; Groover, 2007). However, in real world manufacturing it is important to include some overrun, which is a path that the tool must travel to enter and exit the part. This is referred to as *approach and retraction* (Black and Kohser, 2012). Including this in equation 2.3 gives,

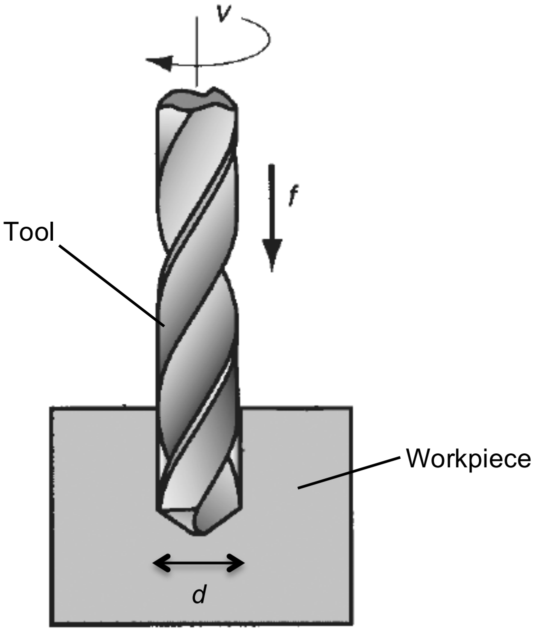
$$t_m = \frac{L+i+o}{f_r} \qquad (2.4)$$

where i and o are the lengths of the lead-in and lead-out travel distances. The material removal rate (R_m) in mm^3/min, is the rate at which a tool removes volume of material and is given by,

$$R_m = vfd \qquad (2.5)$$

where v is the cutting speed in mm/min, f is feed distance in mm and d is the depth of cut in mm.

Figure 2.3. An illustration of a drilling operation. Showing the cutting speed, v, the rotational speed of the tool; the feed, f, the distance moved into the workpiece in one rotation and the diameter of the tool, d. Taken from (Groover, 2007) and adapted.



2.1.2 Drilling

A large number of manufactured products will have holes in them. These are typically used for assembly, to allow materials to be joined together with fasteners such as screws. Other common uses are for weight reduction, aerodynamics and ventilation (Kalpakjian and Schmid, 2008). Drilling is the industrial process used to produce these holes. In general, a rotating tool is fed into a stationary workpiece to create a hole equal to the diameter of the tool. One of the most common machines used for this operation is called a drill press (Crowson, 2006).

A drill uses a pointed tip to cut into a surface. Two spiral flutes run up the length of the drill, which allows material to pass up to the top of the surface (Kalpakjian and Schmid, 2008). The cutting speed in drilling is specified as the surface speed at the outside diameter of the drill. The rotation speed, N, in revolutions/min is given by,

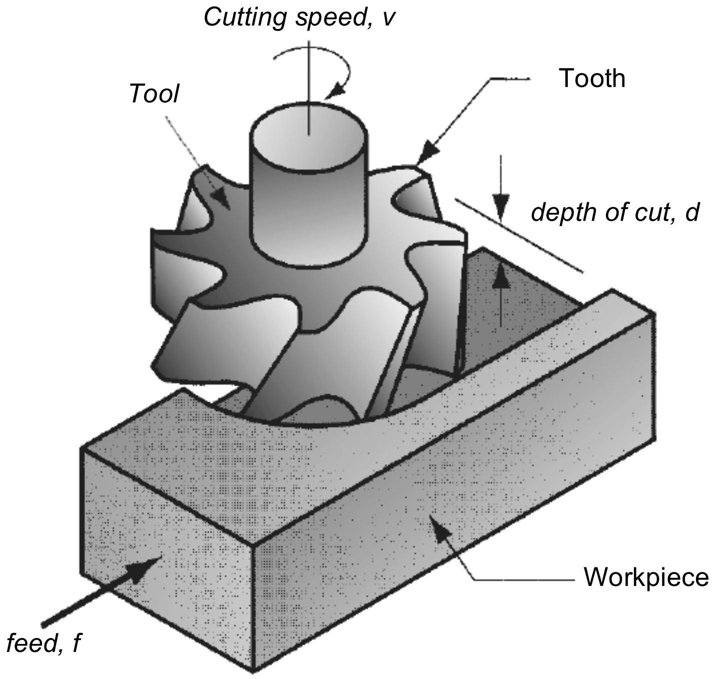
$$N = \frac{v}{\pi D}$$

where D is the drill diameter in mm and v is the cutting speed in mm/min. The feed is the distance moved into the part in a single revolution measured in mm. The feed rate in mm/min is given by,

$$f_r = fN$$

The equations for both N and f_r are identical to those used in turning, described in equations (2.1) and (2.2) above.

Figure 2.4. Illustrating an end milling operation. Showing the depth of cut, *d*, the distance that the tool penetrates the workpiece; the feed, *f*, the distance that the workpiece moves during every revolution of the tool; and the cutting speed, *v*, the speed that the tool rotates. Taken from (Groover, 2007) and adapted.



2.1.3 Milling

Turning and drilling are fundamentally used to create cylindrical profiles, be it external (turning) or internal (drilling). Milling uses similar techniques to these processes but can produce a huge variety of shapes. Complicated geometrical forms can be produced to extremely close tolerances (under 5 microns) with a fine surface finish (Black and Kohser, 2012).

In milling, a rotating tool with multiple cutting edges, known as *teeth*, has a workpiece fed against it. There are two main types of milling operation, peripheral and face. In peripheral milling the axis of the spindle is parallel with the part. This provides for many useful operations. In slot milling, vertical slots are cut into a part by a cutting tool narrower than the workpiece. Saw milling is similar but the cutting tool is used to cut the material in half. Another common operation is slab milling, where the tool has a diameter wider than the workpiece allowing fast machining of broad surfaces.

In face milling, a cutting tool is connected to the end of a spindle. The axis of rotation of the spindle is perpendicular to the work surface. As with peripheral milling, there are many different operations. Conventional face milling is similar to slab milling, where the diameter of the cutting tool is larger than the workpiece. In profile milling, the cutter is on the outside of the part, cutting the surface from the side (Groover, 2007).

A very versatile form of face milling is the end milling operation. This is used for all of the optimisation work in this thesis. Here, the cutter attached to the end of the spindle has a diameter smaller than the workpiece. By using different shaped cutters, a variety of profiles and curved surfaces can be created. Kalpakjian and Schmid write that,

> "End milling can produce a variety of surfaces at any depth, such as curved, stepped and pocketed." (2008:668)

This shows the flexibility of the operation. It can be used to produce a huge number of different geometrical forms with some example applications being aerospace components, intricate dies and moulds and submarine propellers. Cutting conditions in milling are similar to those in turning and drilling. Cutting speed is determined using the outside diameter (in mm) of the milling cutter, as in drilling:

$$v = N\pi D \qquad (2.6)$$

where v is in mm/min. Unlike turning, the cutter used in milling almost always has multiple teeth, which means that a different calculation has to be made for the feed rate. As the workpiece moves against the cutter, the feed rate is also called the *table feed*, or the *linear travel rate*. In mm/min, the feed rate is given by,

$$f_r = Nzf_z \qquad (2.7)$$

where N is the rotational speed in rev/min, z is the number of teeth on the milling cutter and f_z is the feed per tooth – the distance travelled a single tooth in one revolution. The metal removal rate, in $\mathrm{mm}^3/\mathrm{min}$ is given by equation

$$M_r = Ddf_r \qquad (2.8)$$

where D is the diameter in mm, d is the depth of cut in mm and f_r is the feed rate in mm/min. The total machining time, is given by,

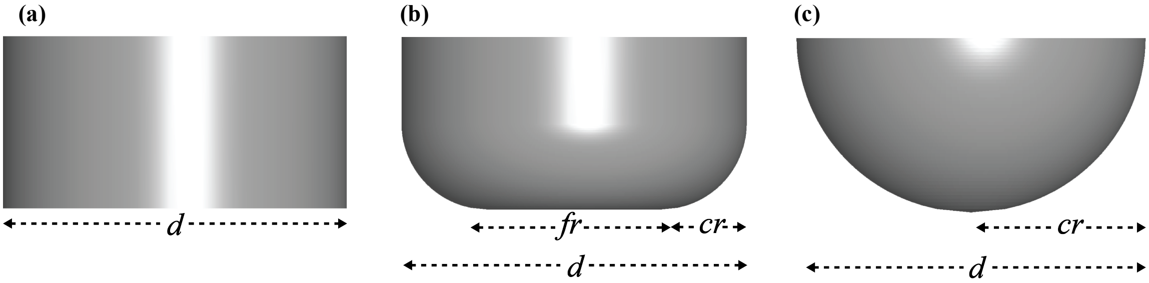
$$t_m = \frac{L}{f_r} \qquad (2.9)$$

where L is the total linear travel distance of the cut in mm. Another important segment of the cutting operation in real world milling are lead-in and lead-out motions. These are circular paths that a cutter follows in order to enter and exit a part cleanly (Smid, 2003). This adds to the total distance travelled by the cutter. Finally, rapid motions are used to move the cutting tool around the part when the tool is not in contact with any material. An example of an end milling operation is shown in Figure 2.4.

2.1.4 Cutting Tools used in Milling

The most common types of tool used in end milling are the flat end mill (normally referred to simply as end mill), ball nose and toroidal (also known as bull nose) cutters (Smid, 2003). An end mill has a flat radius that is equal to the diameter of the tool, and is best used to create flat surfaces. Toroidal tools have flat ends with a corner radius that is less than half the diameter. This means that a portion of its end is curved and the rest is flat. This enables it to cut flat surfaces that require a corner radius between the wall and the surface. They can also be used to produce some sculptured surfaces. Ball nose cutters have hemispherical ends and can be used to create curved and sculptured surfaces (Smid, 2003; Kalpakjian and Schmid, 2008). Illustrations of the three tool types can be seen in Figure 2.5.

Figure 2.5. Showing the diameter (*d*), corner radius (*cr*) and flat radius (*fr*) for the 3 main tool types used in machining. The tools are (a) end mill (b) toroidal and (c) ball nosed.



If all the tools had the same cutting diameters and speeds, the end mill would be able to remove the most material in the same amount of time. However, it can only make cuts that are as narrow as its cutting diameter. The ball nosed cutter, with its fine tip, is able to make much smaller cuts than its diameter, which means that it can achieve a lower surface tolerance than the end mill. The geometry of the cutter gives it this advantage but also means that the ball nose struggles with cutting a horizontal planar surface, which the end mill and toroidal cutters are much more successful at.

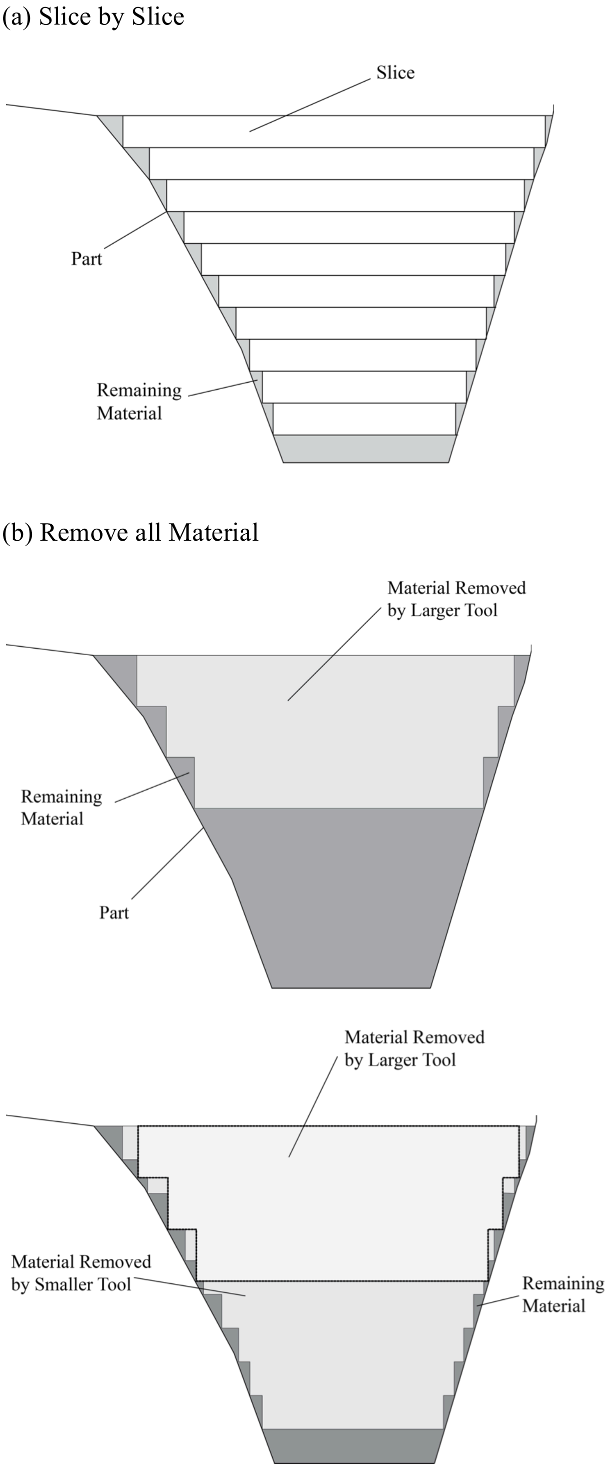
2.1.5 Roughing and Finishing

In machining, particularly with turning and milling, the operation is normally divided into *roughing* and *finishing* cuts. Roughing is applied to remove the bulk of the material quickly, in order to reduce the workpiece to a surface tolerance where the finishing procedure can be carried out. The finishing cut is used to ensure an accurate dimensional tolerance and the desired surface roughness of the final part (Groover, 2007; Kalpakjian and Schmid, 2008).

In the literature reviewed below, two different methods are used for roughing. The first is the 'slice-by-slice' or 'layer by layer' method, which, for example, is used in (Vosniakos and Krimpenis, 2002). The component is divided into slices, of a size determined by the process planner or planning system. A tool is chosen to remove all of the material that it is able to cut in a particular slice. The tool that can be used in each slice is restricted by the geometry of this slice. If the tool has too large a diameter it will gouge the sides of the part and remove too much material, distorting the desired shape of the part. The volume in which a tool can safely remove material is called the *accessible region*, and the total desired volume of removal is the *target region* (D'Souza et al., 2004). While tool changes can occur at each slice, to save time consecutive slices can use the same tool, resulting in the layers being merged together.

A second method is found, among others, in the work of (Lim et al., 2001), (Ahmad et al., 2010) and (Chen and Fu, 2011). This is the method used in the experiments in chapters 4 – 9 in this thesis, and is the preferred method in industry (Steve Youngs, *Vero Software*, Personal Communication). Here, a tool sequence is chosen and a tool path is generated for each tool to machine the maximum amount of material to the extent of its geometrical constraints. This technique allows for a larger tool to machine to a greater depth than it may be able to with the slice-by-slice method. A main reason for this is to lower the risk of tool breakages. The first tool used will be the largest in the sequence. Larger tools are normally short, which combined with their larger diameter, makes them stronger. It is therefore

Figure 2.6. Illustrating (a) the slice by slice method and (b) the remove all possible material roughing techniques.



(a) Slice by Slice

(b) Remove all Material

Slice
Part
Remaining Material
Material Removed by Larger Tool
Remaining Material
Part
Material Removed by Larger Tool
Material Removed by Smaller Tool
Remaining Material

desirable to remove as much material as possible with these tools, even if they can only remove a small part of a horizontal slice. The two roughing methods are illustrated in Figure 2.6.

Roughing operations are the subject of the optimisation work in this dissertation. This is because they are used to remove the bulk of the material, and as a result, improving the performance at this stage is crucial to reducing total manufacturing time. Experiments carried out in (Vosniakos and Krimpenis,

2002) suggested that the time needed for roughing operations were one to two orders greater than in finishing.

2.1.6 Computer Numerical Control and CAD/CAM

Computer Numerical Control (CNC) is the computational process of controlling machine components through programmed instructions. The first prototype was developed by MIT in 1952 (Groover, 2007). A great deal of information can be transmitted to the machine, such as co-ordinates, paths, feeds and spindle speeds. This allows large portions of the machining process to be automated. CNC provides for the ability to accurately produce complex shapes with repeatability.

Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) are an integral part of the modern workshop. CAD is used to design and model 3D geometrical forms (Black and Kohser, 2012; Kalpakjian and Schmid, 2008). They are used to create the geometrical specification for a component. CAM is defined by the U.S. Congress Office of Technology Assessment as,

> "Those types of programmable automation which are used primarily on the factory floor to help produce products." (US Congress, 1984: 48)

This description is necessarily vague as CAM can apply to several different areas in automated manufacturing. In machining, a common application of a CAM system is to take a 3D model of a component and a set of cutting parameters such as tool geometry and depth of cut, and generate the tool paths used to cut the component. These can then be converted to CNC instructions and sent to a machine to perform the operation and produce the part.

2.1.7 Optimisation in Computer-Aided Manufacturing Systems

Flexible manufacturing systems, which use highly automated computer-controlled machines, are extremely expensive to purchase and maintain. Therefore, it is important that they run as efficiently as possible. Jobs need to be finished as fast as they can and down time, when no job is being processed, needs to be kept to a minimum. There are a number of areas in which optimisation can be performed. Firstly, the schedule of operations that will produce all the features of a part can be optimised. Broadly this is called process planning, and computational methods are referred to as *Computer-Aided Process Planning* (CAPP) (Kalpakjian and Schmid, 2008). CAPP involves the selection and sequencing of operations required to manufacture single or multiple components under the constraints of available resources in a factory or multiple factories (Salhei and Tavakkoli-Moghaddam, 2009; Zhang et al., 1997). An example of this is *Job Shop Scheduling* (JSS), which is the assignment of jobs to machines in order to satisfy objectives such as maximum production time or minimum downtime (Davis, 1985).

Another way to improve manufacturing efficiency is to optimise the individual operations. Machine tool parameters such as tool geometry, depth of cut, cutting speed and feed-rate are largely chosen by human experts, guided by their experience and information found in handbooks (Venkata Rao and Pawar, 2010). The strategies recommended by the handbooks are rarely optimal and as such leave much room for improvement. An example of this is found in (Baskar et al., 2006), who found that a Genetic Algorithm gave a 400% increase in profit rate compared to handbook recommendations.

2.1.8 The Tool Selection Problem

The focus of this thesis is to solve the *Tool Selection Problem* in machining. This will be specifically applied to rough end milling operations, to produce a wide range of complex geometrical forms. This can be seen as a halfway house between a *CAPP* and Machining Parameter Optimisation. Let us start with a simple example. A Roughing operation is required in a workshop on a machine that has an automatic tool holder containing two different tools. The machine is set up with a blank workpiece, which by the end of the operation must be close to a desired final geometry. Specifically this closeness must be within a specified tolerance. Our set of tools contains one with a large diameter, while the other has a small diameter. Without repetitions, which are redundant as a tool will remove all the material possible in its first run, there are four possible permutations available here:

1. **Large**
2. **Small**
3. **Small – Large**
4. **Large – Small**

If we assume that the two tools have the same cutting speed, of these four strategies, number one will be able to remove the largest amount of material in the shortest amount of time. However, it may not be able to get within the desired surface dimensional tolerance, as it could be too large to reach some areas. The small tool may be able to get within this stopping tolerance but will clearly take a longer time to do so, as its smaller diameter means it will take smaller cuts. In this scenario therefore the best solution is likely to be number four, which will allow the large tool to remove the bulk of the material and the small tool to remove the remainder before reaching the stopping tolerance.

This example illustrates the problem that is investigated throughout this work: searching for the combination of tools that will remove material to a desired tolerance that minimises an objective or set of objectives. Common objectives are total machining time or total tooling costs. Optimisation of this problem is not handled in current CAM software. An operator will typically design the sequence based on their experience and various rules of thumb (Vosniakos and Krimpenis, 2002).

2.2 Evolutionary and Stochastic Optimisation

Optimisation in manufacturing can be a difficult task. In the case of continuous machining parameter optimisation, the frequent presence of non-trival constraints can introduce many non-linearities (for example if a tool breaks it can add a large amount of extra machining time). This can make problems hard to solve empirically, or to apply traditional mathematical optimisation techniques. In general, scheduling problems are NP-hard, which means that no known algorithms are able to provide an optimal solution in polynomial time (Husbands, 1994). Manufacturing problems often involve huge search spaces, for example a Job Shop Scheduling problem can have 10^{60} combinations, which is larger than the number of particles in the universe (Husbands and Mill, 1991). This makes it impossible to iterate over using traditional search techniques.

Non-traditional bio-inspired techniques such as *Genetic Algorithms* (GA) and *Particle Swarm Optimisation* (PSO) are able to traverse complex search spaces in a global fashion and find good solutions to problems that would be difficult to solve using traditional techniques. Collectively, these methods are known as metaheuristic algorithms, which are defined as general search strategies that use stochastic methods to attempt to efficiently explore a problem space to find (near) optimal solutions (Blum and Roli, 2003; Luke, 2013). In the literature related to optimisation in machining, which is reviewed below, a large number of researchers have utilised metaheuristics on Computer-Aided Manufacturing problems. Many of these employ *Evolutionary Computation*, which is a term that describes Genetic Algorithms and related techniques.

JSS, the Tool Selection Problem and Machining Parameter Optimisation are global optimisation problems. In a global optimisation problem, the goal is to find the set of variables that represent the globally minimum output of a function. If **S** is the search space, f is an objective function and $g_i\colon \boldsymbol{S} \to R^n, i \ \in \{1, \dots, q\}$, is a set of q constraint functions, then the global minimisation problem is to:

Minimise $f(x)$, such that $g_i(x) \leq 0 \ \forall i \in \{1, \dots, q\}, x \in S$ (2.10)

At a point $x \in S$, a constraint, g_i is

Satisfied, iff $g_i(x) \leq 0$;

Violated, iff $g_i(x) > 0$; (2.11)

The subset $g_i(x) \leq 0 \ \forall i \in \{1, \dots, q\}, x \in S$, the set of points x that satisfy all of the constraints, is known as the *feasible* region (Watanabe and Hashem, 2004).

Stochastic Optimisation refers to the use of random methods to move between points in decision space, that represent solutions to an optimisation problem. Stochastic search is useful for problems with large search spaces that are arbitrarily complex, and where it is difficult to obtain a gradient or derive the objective function mathematically (Watanabe and Hashem, 2004; Goldberg, 1989). A large number of stochastic optimisation algorithms are naturally-inspired and commonly used techniques include: *Genetic Algorithms* (GA), *Simulated Annealing* (SA), *Evolutionary Strategies* (ES), *Particle Swarm Optimisation* (PSO) and *Ant Colony Optimisation* (ACO). The focus of the work in this thesis is applying Genetic Algorithms to CAM problems. Genetic Algorithms were chosen because of their application to a wide range of disciplines, their ability to handle rugged non-linear search spaces and constraints and easy parallelisation. The Tool Selection Problem is combinatorial in nature, while machining parameter tuning has continuous properties. Genetic Algorithms have been shown to perform well in both of these domains (Watanabe and Hashem, 2004), while for example; PSO and ES are primarily used for numerical problems. For these reasons, only Genetic Algorithms and their simpler relative, *Hill Climbing*, will be discussed in detail below.

Algorithm 2.1: Pseudo-Code for the basic Hill Climbing algorithm

```
x = random(X)
while(stopping_condition == False), do
    x' = choose(x,X)
    if fitness(x’) < fitness(x), do
        x = x’
    end if
end while
return x
```

2.2.1 Hill Climbing

The Hill Climbing (HC) algorithm is a simple stochastic optimisation algorithm that can be used for both combinatorial and continuous optimisation problems. It is related to the Genetic Algorithm, and so serves as an introduction to the fundamental concepts. A basic implementation can be found in the pseudo-code in Algorithm 2.1.

There are many varieties of HC algorithm. A common formulation is known as Stochastic Hill Climbing (Forrest and Mitchell, 1993). Here, the choose(x,X) function will randomly select a new solution, x', that is in the local neighbourhood of x. For example, in a continuous problem, the neighbourhood may be all values in the range:

$$[x - 0.1, x + 0.1]$$

This new solution, x', can be seen as a mutated copy of x. From a minimisation perspective, which will be the default in our optimisation discussions, if the new solution has a smaller objective value (fitness) than the current best it replaces it. This procedure is repeated until a stopping condition, such as a pre-specified maximum number of iterations, is reached. In the *Steepest Ascent Hill Climbing* (Russell and Norvig, 2003) formulation, the choose(x,X) function surveys all the neighbours of x (e.g. solutions one bit flip away) and returns the solution that represents the greatest improvement on x (Forrest and Mitchell, 1993). This is analogous to the calculus-based gradient ascent/descent method (Goldberg, 1989). In its basic form, HC is a local search strategy. It searches for improvements to a solution in its local neighbourhood. This means that it is susceptible to getting stuck in areas of local optima. For example, if the objective function has two peaks, the algorithm will determine that the top of the first peak is the global optimum (Goldberg, 1989). A related algorithm is the (1+1) Evolutionary Strategy (Beyer and Schwefel, 2002). Here, at each iteration there is a current solution and a generated mutated copy. The solution is represented in such a way that it can be mutated to represent any other solution in the search space. For example, if a discrete search space contains exactly 8 solutions they can all be represented using a 3-bit binary string. An example mutation method, which would correspond to the choose(x,X) method in Algorithm 2.1, could first create a copy of the current solution. Then with a certain probability, each bit in the 3-bit sequence could be flipped. This would enable the current solution to be transformed into any of the possible solutions in the search space and importantly, moves the algorithm from a local to a global optimiser. If the probability of a bit flip is small, then the

Algorithm 2.2: Pseudo-Code for a simple Genetic Algorithm

```
population = []
while population < pop_size, do
  individual = generateRandomIndividual()
  population.append(individual)
end while
evaluate(population)
while stopping_condition == false, do
  temp_population = selection(population)
  for individual in temp_population, do
    if random() < crossover_rate, do
      parent = choose(temp_population)
      recombination(individual,parent)
    end if
    if random() < mutation_rate, do
      mutate(individual)
    end if
  population = temp_population
end while
return population
```

algorithm is likely to return solutions in the local neighbourhood, which makes it similar to the Stochastic Hill Climber with the ability to avoid getting permanently trapped in local optima. Another difference between Evolutionary Strategies and Stochastic Hill Climbing is the use of mutation strength (Beyer and Schwefel, 2002). Here, if mutated copies of a parent have not been showing improvement, the amount of mutation can be increased. For example, if the mutation is on a vector of real values, a mutation function may add a random value drawn from a Gaussian distribution with mean 0 and standard deviation σ^2. In a search run, if it judged that progress is too quick, we may be heading towards a local optimum and so we increase σ^2. Conversely, if we are not making enough improvement we can reduce σ^2, to concentrate on the local search region.

2.2.2 Genetic Algorithms

Genetic Algorithms (GA) were first developed by John Holland and his students at the University of Michigan in the 1970s (Mitchell, 1999). GAs use a model of Natural Selection to search for structures that represent good solutions to a problem. A key difference from Hill Climbing is the use of a population of points (or solutions). This enables the GA to travel through many regions of the search space concurrently. A GA is characterised by having a population of solutions, called *individuals*. Each individual represents a solution by coding the solution into a *genotype*. The decoded realisation of the individual's genotype is its *phenotype*. GAs explore the search space using a process of *selection*, *recombination* and *mutation*. They have been shown to perform well on a number of real world applications, with examples as diverse as Water Distribution Networks (Savic and Walters, 1997), Concert Hall Acoustics (Sato et al., 2002) and Medical Diagnosis (Llorà et al., 2007).

Let us start with an example of a simple GA to explain the concepts. The algorithm is initialised by creating a population of randomly generated genotypes. If the genotypes are bit strings then they could be generated by implementing a coin flip to choose each bit in the string. After the population is generated, the phenotypes are evaluated according to an objective function known as the *fitness function*. This gives us the first generation of the population. To create the second generation, individuals are copied from the current population to form a new temporary population. Which individuals are copied and how many of each is determined by a selection method. A common method is to probabilistically select individuals, where the chance of selection is proportional to their relative fitness. Once the temporary population has been created, variation operators are probabilistically applied to each individual. The recombination operator, operating with a pre-specified probability known as the *crossover rate*, chooses two individuals and swaps elements between the two genotypes. The mutation operator, which occurs with a probability specified by the *mutation rate*, applies a random change to the genotype of an individual, such as a bit flip at a random point in the bit string. The new population is then evaluated and the process repeats until a stopping condition is met (Goldberg, 1989). This process can be seen in Algorithm 2.2.

2.2.2.1 Genetic Algorithms and Traditional Techniques

Traditional optimisation methods can be split between calculus-based, enumerative and random. Calculus-based methods require the objective function to be differentiable, which can be difficult to achieve in real world problems (Goldberg, 1989). As with the Hill Climbing example, they are also susceptible to discovering local minima because they cannot discern between locally and globally optimal peaks - they discover all minima.

Enumerative methods and many classical search techniques attempt to look at all of the solutions in a search space, which may or may not be bounded. Many search spaces are too large to survey in this manner, while it can be difficult to create a divide and conquer strategy or provide a good heuristic function to direct search. Through the GA's parallel global search methods, it can find good solutions in large spaces by surveying only a fraction of the total number of solutions. However, there are certain search problems where traditional search, such as tree traversal, is likely to out perform a GA. For instance, in problems where the path to the goal is important rather than the generation of the final solution, such as in the 8-puzzle problem (Mitchell, 1999).

GAs work directly with the output of objective functions and therefore do not require derivatives to guide the search (Goldberg, 1989). They also do not require heuristics or knowledge to be known about the search space in advance, although knowledge is used to design the fitness function and genotypic representation. This enables them to be applied to a wide range of complex problems. They are classified as meta-heuristics as they use heuristics that they obtain online through sampling solutions. If heuristics are known, they can also be used with GAs (Watanabe and Hashem, 2004), and hybrid techniques have been shown to perform very well, such as in the Travelling Salesman Problem (Nguyen, 2007). However, in a situation where there is a well understood problem with strong domain knowledge, a heuristic or expert system is likely to outperform a GA (Mitchell, 1999). Evolutionary Algorithms are also known to be robust (Goldberg, 1989; Watanabe and Hashem, 2004). This means that over different runs, the GA will tend to return similar results. Pure random search may be able to find equally good solutions to a GA but there will be much greater variation in results. Additionally GAs are able to take advantage of structural regularity in search spaces that contain this, which random search ignores.

Another advantage of GAs is that they are naturally parallel (Cantú-Paz, 2000). As they work with a population of independent individuals, it is always possible to evaluate the entire population in parallel, meaning that the total run time of the algorithm can be reduced. Parallel Evolutionary Algorithms are described in more detail in Chapter 9, where they are applied to Tool Selection Problems.

2.2.2.2 Design Issues in Genetic Algorithms

The main challenge to a successful implementation of a Genetic Algorithm is premature convergence on suboptimal solutions (Andre, 2001; Watanabe and Hashem, 2004). In a limited amount of computational time, an Evolutionary Algorithm does not guarantee the return of an optimal solution and may not even return a good solution. If a globally optimal solution is required other, enumerative techniques will likely offer better performance. There are many factors that contribute to premature convergence, such as the encoding (representation) of the problem; operator bias in selection, mutation or recombination; parameter values; lack of diversity in the population; and deception.

The coding, or representation of the problem can prevent a GA from reaching the optimum solution by making it difficult to traverse the search space. In general, a small change in the genotype (coding) should lead to a corresponding small change in phenotype (actual solution). Bentley writes,

> "Poorly designed representations are most problematical towards the end of an evolutionary run. As the population converges onto a small area of the search space, fine tuning these solutions becomes nearly impossible if every minor change in the genotype causes a major change in the phenotype." (1999:52)

Similarly, operators such as mutation and crossover can create sizeable changes in individuals, which have a disruptive effect on search. These operators can also introduce search bias. For instance a mutation operator may bias solutions towards certain areas of the search space, which causes the population to converge early.

Another important issue is epistasis. In biology, a gene is epistatic if it masks the phenotypic expression of another gene. In a Genetic Algorithm context, it refers to the interactions between alleles in the genotype. If there is minimal epistasis then all of the genes in the genome are independent of one another (Naudts and Verschoren, 1999). In a situation of maximal epistasis, all subsets of a genome string are related. High levels of epistasis mean that there is complex structure in the search space, which can be difficult for a GA to extract using standard operators. This means that little information can be transferred between individuals in the population.

Deception is a related but different problem. Here, lower order schema (or building blocks) relay information that points away from the optimum solution. For example, if a fitness function is the sum of the ones in a string, except if a string contains all zeros, where the fitness is the sum of the zeros plus one, then the search gradient will be towards a string with all ones and away from all zeroes (Mitchell, 1999). To avoid deception, the fitness function needs to be carefully considered.

Duplication can also create premature convergence in a Genetic Algorithm (Ronald, 1998). In a fully converged population, every member of the population is the same. If higher than average fitness duplicates exist in a population, this increases the probability of further duplicates joining the population in future generations. When there is a fall in the level of diversity, which is the variance in individuals' fitnesses, the GA loses its parallel search capabilities. This is heightened through the process of Genetic Drift (Rogers and Prugel-Bennett, 1999).

One way to avoid premature convergence is to maintain a high level of diversity in the population (Ronald, 1998). This can be achieved in several ways. Having a high mutation rate can introduce new individuals to the population at each generation. However, if this is too high it can have a disruptive effect on search. Fitness sharing and speciation are other techniques that can be used (Mitchell, 1999). They both rely on determining the similarity between members of the population. This may not always be an easy task. In fitness sharing, the fitness of similar individuals is reduced. In speciation, only similar solutions are able to recombine, creating distinct "species". Another method for avoiding premature convergence, first used in parallel computation settings, is the concept of geographical distribution or multi-demes (Cantú-Paz, 1998). Here, multiple separate populations run in parallel. At certain points in the evolutionary cycle, individuals are migrated between populations. However, this can come at an increased computational cost.

2.2.2.3 Constraint Handling

One strength of Genetic Algorithms is their ability to handle non-linearly constrained optimisation problems, which are often present in real world scenarios. This can pose a challenge to classic linear and nonlinear programming methods (Watanabe and Hashem, 2004). There are several ways that constraints can be handled by a GA and other stochastic algorithms (Michalewicz, 1998). The first is to reject infeasible individuals when they are generated, known as the "death penalty". If an infeasible solution is generated, it is thrown away and a new individual is created, repeating the process until a feasible solution is found. This can be computationally expensive if the individual must be evaluated first. Another related method is to have a repair operation. If an infeasible solution is generated, it is modified so that it becomes feasible. In a simple example we could have a constraint that a variable has to be a positive number. If a negative number is generated, it could be set to 1. The repair operation will be problem specific and could be computationally expensive or difficult to achieve.

The above two methods share a problem, which is that information from infeasible solutions is lost. Often, solutions that lie on the boundary of the feasible region are optimal. This can be remedied by relaxing the constraints by a small amount to allow some infeasible solutions in the population. Another method, which is the most widely used in evolutionary algorithms, is the penalty function (Watanabe and Hashem, 2004). Here, a constrained problem is transformed into an unconstrained problem by penalising infeasible solutions. For example, in a minimisation task, an infeasible solution may have a penalty of 1000 added to its fitness value. Creating an appropriate penalty function can be a difficult task. If the penalty is too large, the population will be encouraged to move far away from the feasible/infeasible boundary. If it is too small, the population may converge on infeasible solutions. The difficulty with both penalty functions and repair operations is that they distort the search space, which can bias search and lead to premature convergence. Their implementation must be carefully considered.

2.2.3 Multi-objective Optimisation

Another feature of Evolutionary Algorithms, which will be taken advantage of in this thesis, is their ability to perform multi-objective optimisation (MOO). In a single-objective optimisation task, the goal is to find a single solution that optimises an objective function. A simple example would be to maximise the number of ones in a bit string. In MOO, there are several objective functions, such as time and cost. To extend the example above, in addition to maximising the number of ones, a separate objective could be to maximise the number of “01” pairs in the string. These objectives are conflicting. If both are equally important then there are many optimal solutions, which represent different trade offs between objective functions.

A common way this is handled in optimisation problems is to combine the objectives into a single function. Prior to optimisation, each objective can be given a weight based on its relative importance, which allows a single solution to be found at the end of search (Coello Coello and Lamont, 2005). In MOO, each objective function is considered to be equally important, and the goal is to find a set of solutions representing optimal trade-offs between objectives. A solution is considered to be Pareto optimal if no possible change to the decision variables of a solution exists that will provide an improvement in any one objective function without deterioration in the values of any of the others. This allows the decision maker to make an informed selection post hoc, and so search does not need to be biased by a priori weightings. A detailed review of multi-objective techniques is given in Chapter 6.

2.3 Optimisation in Machining

In Section 2.1 an overview of machining operations was presented, while in Section 2.2 the reader was introduced to the main concepts, advantages and challenges faced by evolutionary and stochastic optimisation. This enables us to move to the motivation behind the work in this thesis, which is to contribute to the field of Evolutionary Computation applied to machining operations. The main focus here is the lack of support in current CAM software to suggest combinations of tools that offer good solutions to machining problems. This means that this job is left to the machine operators, who effectively need to solve the problem by hand (Vosniakos and Krimpenis, 2002).

While our focus is on finding good tool sequences, which is explored in the experimental work in Chapters 4, 5, 7 and 9, another important area that has been addressed in the literature is Machining Parameter Optimisation. Interestingly there has been a divide between Tool Selection and Machining Parameter Optimisation research. With very few exceptions (for example, Krimpenis and Vosniakos, 2009), researchers have concentrated on one or the other. The experimental work presented in Chapter 8 adds to this small body of work and is the first to use a multi-objective approach. In the following two sections we will first review related work in Tool Section Optimisation before discussing key work in the Machining Parameter Optimisation literature.

2.3.1 Tool Selection Optimisation

A number of researchers have taken a geometrical approach to tool selection. In (Lin and Gian, 1999), this is applied to slice by slice roughing to find tool combinations that can remove material to within a certain tolerance, producing a part with a sculptured surface. A 3D mesh of the part is inputted to their system, which calculates a tool sequence and generates tool paths. Their approach can be summarised as finding the largest tools that are able to machine each slice. Once a set of legitimate tools has been determined for each slice, if a tool has been chosen for consecutive slices, it performs all of these cuts before switching. As the tool path is produced after the selection decision, the actual time taken by each tool is not considered in the optimisation. Furthermore, tools are considered in terms of size only, which could lead to inefficient strategies because the relative speeds that the tools cut at has been ignored. Only end mill tool types are examined and tool wear is also not considered in the optimisation.

Another geometrical-based tool selection optimisation method is presented in (Lim et al., 2001). This is applied to the roughing method of having each tool remove as much material as it can. The optimisation begins by finding the largest tool that can remove the entire volume required to create the component, within a specified surface tolerance. The next tool is then chosen by working out the relative delta-volume clearance rates (RDVC) of the other tools. The tool with the lowest RDVC is selected. This will effectively be a tool that clears a large amount of material in a short amount of time. There could be problems with fixing the last tool based on size only. The relative differences in the cutting parameters of tools mean that a smaller tool could finish faster than a larger tool depending on the length of the tool path. By not considering tool paths, the computational cost of this method is kept very low but this comes at the expense of accuracy. This is exacerbated by other important parts of the tool path such as lead-ins, lead-outs and rapid movements, which increase the time taken by each tool but are not included in the RDVC. It would also be difficult to extend the model to include tool wear. Again, only end mill tool types are considered.

In (Wang et al., 2005a), the machining problem is divided into three processes: roughing, semi-roughing and finishing. Their system allows the use of several different tool types, which makes it more industrially relevant than some of the other work reviewed here. The choice of the tool type is made using a system of rules, which makes a decision based on the geometry of the features in the component. The tool size is selected after the tool type is determined. This could prevent optimal sequences from being found, because it restricts the set of tools considered for each operation. Roughing is completed using the slice-by-slice method. Here, the set of tools able to completely machine each slice is determined. The material removal rate is calculated for each tool using handbook values for parameters, and the total time taken and residual material left are combined to make a single tool selection decision for each slice. Actual tool paths are not used in the selection decision.

Following the completion of the roughing process, a tool is chosen for semi-roughing and finishing. For the former, the smallest tool with a depth of cut larger than the largest 'step' left by roughing is chosen. There is no optimisation here based on tool types, which could affect the machining time. For finishing, the largest tool is chosen that has a radius smaller than the radius of curvature of the component. This work is interesting as it provides a complete solution for the automatic machining of a component but it could be extended in a number of ways. It would be interesting to optimise both roughing and semi-roughing at the same time because the profile left by roughing will affect which tools can be used in semi-roughing and therefore there might be a more efficient solution when both are considered together. Finally, including tool wear would make the optimisation more relevant to real world applications.

Another project that concentrates on 3D pocket machining is found in (Bouaziz and Zghal, 2008). Here, machining is divided into several types of operation, in a similar way to (Wang et al., 2005a). The first is a hollowing out procedure, which they refer to as roughing. Under-roughing is then carried out in order to remove the "staircase", which is left by the roughing operation. A corner machining cycle is applied next to clean up any material that has been left in the corners of the pocket. Finally, a finishing operation is performed to obtain the final shape of the pocket.

Optimisation is applied to determine the best set of roughing and under roughing tools. This works in a similar way to (Lim et al., 2001) discussed above. For roughing, the total machining time is calculated for each tool able to remove the entire material of the pocket. The tool with the least machining time is then chosen and this is fixed. The effect of adding a new tool earlier in the chain is then evaluated and the total machining time is calculated for each of these tools. This is plotted again, and the tool with the lowest total machining time is chosen. This process is repeated for the desired number of tools. A very similar procedure is used for optimising under roughing. This optimisation, while shown to give successful results, could be seen as being limiting as fixing tools in the sequence may prevent more optimal combinations from being found. The system only considers flat end mill tools for roughing and ballnose tools for under roughing. This may prevent optimal sequences from being found. As with other projects, the problem of tool life is ignored.

(Carpenter and Maropoulus, 2000a and 2000b) provide an expert system for optimising tool selection on an array of different milling operation geometries. Here, users are required to explicitly choose the geometry of the operation, and are unable to produce free form sculptured surfaces. For example, the user can choose a T-Slot or a square shoulder, and then supply the system with the specific geometrical conditions of the part. A large number of tool holders and inserts are stored in a database and have classes associated with them which define their suitability to work on types of operation such as a T-slot. When the system selects a tool for a particular operation, the first step it takes is to limit the tool library to tools that have the correct class for the operation. The set of tools are then filtered by their applicability to the geometry of the part, to make sure that the cutter is within the critical dimensions of the part.

Once the final set has been determined, cutting parameters are chosen for each tool. These are based on handbook values and a user-guided procedure, which determines how aggressive the cutting data should be in terms of the range of recommended parameters, which has an impact on tool life. The evaluation of each tool is based on four criteria: maximum metal removal rate, maximum tool life, minimum overall cost and minimum overall time. The user is able to give a weighting to each of the criteria to adjust their relative importance. For example, metal removal rate might be given a high weighting and minimum overall cost a low one. A tool is selected for an operation by calculating the weighted sum of the criteria and then sorting the suitable tools based on this value. While multiple tools can be chosen when there are many operations, this system doesn't allow for more than one tool in a single operation, which can lead to inefficient strategies.

D'Souza et al. (2004) use a graph based method to find the optimal tool sequences needed to perform 3-axis rough machining on free-form pockets. The optimisation goal is to reduce overall manufacturing cost, which is the sum of tool costs, which considers tool wear, and workshop costs, which is based on total machining time. Each tool in the sequence machines to the full extent that it is able to without gouging the sides. As only end mill tools are considered, the geometry of the remaining material after each tool is used is independent of previous tools in the sequence. This means that the tool path of the second tool in a tool pair stays constant regardless of the preceding tool. A directed graph is created by analysing the cost of travelling between groups of tool pairs. This is then evaluated using Dijkstra's Algorithm. In later work presented in (Ahmad et al., 2010) and discussed below, it was found that including tool holders violated the geometrical independence and the authors moved to a Genetic Algorithm for search.

The system is able to find optimal results in a relatively short amount of time, when tested on a library of only 5 tools. However, the project suffers from a few limitations. In creating the tool library, the smallest tool that is included is assumed to be the only tool that can achieve the desired final tolerance. This means that it is fixed at the end of every sequence. This could result in suboptimal tool sequences, depending on whether a workshop owns different tools that can meet this requirement. Another limitation is that only flat end mill tools are considered. Including several tool types could complicate the algorithm because the interaction between different tool geometries may mean that the cost of graph edges can change.

In (Vosniakos and Krimpenis, 2002) the first investigation into the use of GAs for optimising rough milling is presented. The search goal is to minimise the machining time and have the least material remaining after roughing, when tackling a hemisphere-shaped part. The two objectives are combined in a weighted aggregate function. The slice-by-slice method is used, and the GA is tasked with determining the number and height of the slices, and the tool to use on each one. For each slice, when a tool is evaluated, the algorithm generates its own approximate tool paths, which allows machining time and remaining material to be calculated. The algorithm requires around 3,000 evaluations to achieve optimal sequences and is successful on this task. A limiting factor in this work is that only end mill tools are considered. By not considering other types of tools (tool geometries), which are often owned and used by manufacturers, many efficient tool combinations are ignored and the surfaces available for machining are limited. Tool wear is also not considered here, which is a consideration in real world manufacturing.

Another GA approach to roughing is presented in (Chen and Fu, 2011). Here, the goal is to optimise the selection of cutters for *Aggressive Rough Machining*, which is defined as removing material with total immersion of the tool. Here, each tool machines in one pass (slice). The tool paths are generated directly in the system using a new method introduced by the authors. The objective of the optimisation is to find the set of tools that can cover the maximum area inside the specified geometry of the part, without gouging the sides. This does not consider cutting speed or lead-ins and lead-outs, which are very important for real world milling. This means that overall machining time is not included in the optimisation, and so sub-optimal solutions may be found, depending on the relative differences in the tools. Only end mill tools are included, and so other tool types are ignored, which are regularly used in an industrial setting.

Unlike most of the work reviewed here, (Ahmad et al., 2010) explicitly include tool holder geometry in their milling optimisation. This builds on the work of (D'Souza et al., 2004). They argue that many projects optimise tool selection by concentrating on the geometry of the cutter but ignore the tool holder. This leads to a situation where tool path verification needs to be carried out to find collisions between the tool holder and the component. An engineer or process planner will then have to change the sequence by trial and error in order to find a working solution. They also state that the assumption in (D'Souza et al., 2004) that a larger tool is the geometrical subset of a smaller tool can be violated by tool holder geometry.

In their work, the authors used a GA to optimise the tool sequence of both the tool and tool holder specifically for 2.5D pocket machining. The algorithm was limited to optimising for one pocket at a time, and the pocket had to be explicitly defined. Each tool machined the full area accessible to it, which is the method prevalent in industry. As with their previous work, a 'critical tool' is fixed at the end of each sequence. This is the largest tool that is able to remove all of the material within a defined surface tolerance. This is based solely on geometrical conditions, so could result in sub-optimal sequences. The results show that the GA is able to find good solutions at the end of the evolutionary search. The authors use a fitness function that is a combination of total machining time and manufacturing costs. It would be interesting to use a multi-objective formulation, which could show trade-offs between the two objectives. The research also only considers end mill tools and the presence of other tool types may cause problems with the assumptions used.

(Krimpenis and Vosniakos, 2009) used a GA based search method to find a single tool and its associated cutting parameters to optimally rough a component using CAM software. Every parameter available to the user in the software is included in the optimisation process. These include standard parameters such as cutting speed, feed rate and depth of cut. The authors identified three objectives, to minimise the machining time, maximise removed material and maximise the uniformity of the remaining volume at the end of roughing, creating a three dimensional Pareto front.

Two Genetic Algorithms are used, a GA with a small population (referred to as a micro-GA) and the Strength Pareto Archiving Evolutionary Strategy 2 (SPAES2; Zitzler, Laumanns and Thiele, 2002), which is a multi-objective algorithm. The primary GA determines the number of slices used in the procedure and the other cutting parameters. The secondary GA determines the height of each slice. Each solution (set of parameters) is evaluated using a simulation provided by the CAM software. The system is able to find very good results for each of the three objective functions. However, as only a single tool is selected and tuned, it cannot create an efficient tool plan. Tool wear is included as a technological constraint but it would be interesting to add it as an objective.

(Spanoudakis et al., 2008) applied a GA to both tool selection and the optimisation of machining parameters on a 3-axis roughing problem. Analysing the effect of changing various cutting parameters, they found that the depth of cut is the most important affecting both material removal volume and total machining time. Tools remove all of the material that they are able to. The choice of tools and their associated parameters are optimised using a differential GA. To evaluate results, solutions are simulated using CAM software. Results from the simulation are compared to a real life milling operation and the simulation is found to be very accurate.

The optimisation task is restricted to select a single roughing tool, and a single finishing tool. This means that sequences are fixed at two tools in length, which could be a very large limitation in real world applications. The sequences are evaluated using an objective function that weights the total machining time and volume of remaining material. A tool library of tools is available including four flat end mill and four ball nosed tool types. This could be improved by also including toroidal tools, which are very prevalent in industry. As with many other projects, the authors do not include tool wear in their optimisation.

2.3.2 Machining Parameter Optimisation

The final two systems discussed above combine tool selection and parameter optimisation, albeit in a restricted manner. While this is also tackled in a very limited way by (Wang and Jawahir, 2005), there has been a large body of work concerned with Machining Parameter Optimisation of a single tool, which mainly focuses on the three important machining conditions: cutting speed, feed and depth of cut. A selection of relevant work is presented below, with an emphasis on studies taking a multi-objective approach.

As recently described, (Wang and Jawahir, 2005) include elements of tool selection in a multi-pass turning optimisation system where the cutting speed, feed and depth of cut are optimised. They use an objective function based on a weighted aggregation of surface roughness, cutting force, chip form and tool life. The weightings are chosen arbitrarily and it is up to the process planner to decide whether to emphasise surface roughness or tool life for example. Interestingly, total machining time is not a consideration here. Constraints are dealt with using penalty functions. Experiments are carried out on a two-pass and three-pass turning operation, where either surface roughness or tool life are given higher weightings. The GA is found to converge on good solutions. It is clear that a multi-objective perspective could be beneficial here. It is noted by the authors that the cutting tool itself can have a big effect on the operation. In a further experiment all the four possible combinations of two cutting tool inserts have their cutting parameters optimised by the GA. This is interesting but the selection of the tool is not included in the optimisation, which means that using this system and a library of tools, every possible sequence would have to be evaluated.

Baskar et al. (2006) use a GA, hill climbing algorithm (HC) and a hybrid of the two to optimise cutting speed and feed for a multi-tool milling operation. Maximising the total profit rate is used as the single objective, which is derived from the sale price of the component and the manufacturing costs (including tool wear), over a period of time. The cutting tools and the number of operations are decided in advance. The algorithms are compared to the method of feasible direction, and handbook values. At the end of optimisation, all three algorithms outperformed the handbook values and the method of feasible direction. The GA was shown to perform better than HC and the hybrid the best of all, improving on handbook values by over 500%.

In (Wang et al., 2005b), a parallel Genetic Algorithm hybridised with parallel Simulated Annealing is used to optimise cutting speed and feed rate in a multi-pass milling application. A 37-processor cluster is used to provide coarse-grained parallelisation. A single objective function is used, minimising production time, which considers total machining time and tool wear but does not consider tooling costs. The tool diameter is chosen ahead of time and has a flat end mill geometry. The algorithm is repeated with four different depths of cut. However, including depth of cut as a parameter in the optimisation procedure could create better and more industrially relevant results. The researchers found that the hybrid located the optimal solution much quicker than the GA on its own. The results were compared to geometrical programming and the hybrid was able to find better solutions in shorter amounts of time, for all depths of cut.

This work is extended in (Wang et al., 2006) to use a multi-objective approach to cutting parameter optimisation in milling. Here, the NSGA algorithm is hybridised with Simulated Annealing to optimise the cutting speed and feed rate parameters. The minimum production time and the minimum production cost are used as objective functions, which includes tool wear and tool life consideration. Four depths of cut are allowed, and the search process aims to find the optimal feed rate and cutting speed for each depth of cut, which allows the depth of cut to be chosen post hoc. Again, a 37-processor parallel system is used, to allow a large number of solutions to be evaluated. This enables a large number of Pareto optimal solutions to be found. As with their previous work, further improvements could be made by including depth of cut as a decision variable. It is also likely that search performance could be improved by using the NSGA-II algorithm.

Another multi-objective approach to cutting parameter optimisation is presented in (Sardiñas et al., 2006), which highlights the importance of providing the decision maker with a selection of solutions. The decision variables considered are the cutting speed, feed and cutting depth. Two conflicting objectives are considered, the production rate (total process time), and the tool life consumed in the process. Coello Coello and Toscano's Micro GA (2001) powers the multi-objective search. Parameters are optimised for a single-tool turning operation. At the end of the evolutionary search, 14 Pareto optimal solutions were found. This allows the decision maker to make an informed decision based on factors such as labour cost and tool cost. This model used for evaluation does not allow for complex aspects such as different tool geometries to be used, as actual tool paths and geometrical profiles are not considered.

Solimanpur and Ranjdoostfard (2008) use a Uniform Design Multi-objective Programming technique (Leung and Wang, 2000) to optimise cutting speed, feed rate and depth of cut parameters to find non-dominated (Pareto optimal) solutions representing good trade-offs across three objectives. These are production time, production cost and surface roughness. Tool life is considered in the evaluation of the objective functions. The algorithm is assessed on two single-tool machining tasks and presents solutions that dominated those found by previous researchers. Only single tools are used, so the more complex tool selection problem is ignored. The Uniform Design method uses a series of weighted aggregate functions to direct search towards the Pareto front in a uniformly distributed fashion. The algorithm is evaluated against VEGA (Schaffer, 1985) but it would be interesting if a more in depth analysis of the Pareto front found was compared to a Pareto-based algorithm such as NSGA-II (Deb et al., 2002) using standard performance measures for quality and spread. A large punishment factor is added to each objective function to prevent infeasible solutions, and it would be interesting to know whether this biases search.

In (Datta and Deb, 2009), NSGA-II and Sequential Quadratic Programming (SQP) are applied to optimising cutting speed, feed and depth of cut on a turning operation. The problem is formulated as having three objectives: unit production cost, unit production time and surface roughness. Again, tool wear is included in the objective functions. A simple problem is tackled, removing a set amount of material from a cast steel blank using a single cutting tool. This does not take into account complex geometries, such as those found in a milling task. A population of 200 is employed over 1,000 generations. A methodical approach is taken to testing the Pareto front obtained by NSGA-II. Each objective is optimised using a single-objective algorithm considering only that objective, to find the extreme points of the front. A number of weighted single-objective aggregate functions are used to create an optimisation procedure to check the intermediate points of the front. Finally, the Pareto points found by NSGA-II have local search applied at the end of the optimisation process to see if better solutions could be found. At the end of this testing, it was found that the NSGA-II was able to converge on the Pareto optimal front without needing to utilise local search. This shows that NSGA-II is able to successfully optimise a multi-objective cutting parameter problem.

2.3.3 Summary of Related Research

The work reviewed above shows that optimisation in machining is an area of interest to many researchers. A restriction present in the majority of previous work on the Tool Selection Problem is the consideration of only flat end mill cutters in tool libraries. This is a big limitation because real world workshops have access to many different tool types. Ball nose and toroidal tools are able to remove material from regions that are not accessible to flat end mills and so can achieve better surface tolerances, as well as being able to produce curved surfaces. While ball nose tools have been considered in (Spanoudakis et al., 2008) and (Wang et al., 2005a), toroidal tools have not. The majority of the Tool Selection work also does not address tool wear or total tooling costs, which is an important manufacturing consideration. Another noticeable absence has been the use of multi-objective optimisation. No work has applied multi-objective techniques to Tool Sequence Optimisation.

Conversely, there has been a large amount of work into applying multi-objective methods to Machining Parameter Optimisation. Researchers have mainly concentrated on cutting speed, depth of cut and feed of single tools. The objective functions used have all been related to a combination of time, cost and quality. Of these, time and cost are greatly conflicting. There is great benefit in being able to present manufacturers with a set of solutions offering a trade-off between these objectives, so they can make a choice based on their current requirements.

There has been very little work that has combined Tool selection, Tool Sequence Optimisation and machining parameter optimisation. In (Krimpenis and Vosniakos, 2009), a single tool is selected from a library of eight tools. In (Spanoudakis et al., 2008) fixed length sequences of two tools are optimised from a library of eight. In (Wang and Jawahir, 2005) only two tools in total are considered. This means that the total number of possible tool sequences considered in these papers is very small, making the combinatorial optimisation problem much simpler. No work has been presented on a multi-objective system that performs simultaneous optimisation of machining parameters and tool sequences.

2.4 Thesis Overview

The motivation of this thesis is to improve the automation and optimisation of Computer Aided Manufacturing (CAM) systems applied to machining. Specifically we will apply optimisation to the rough end milling of sculptured surfaces. The experimental work presented in the following chapters is inspired by what has been achieved by previous researchers and aims to address some of the work that is still needed.

Work in Tool Sequence Optimisation has mainly concentrated on tool libraries containing only one tool type, flat end mill cutters. No work has used the three main tools used in industrial milling, flat end mill, ball nose and toroidal cutters. By using sophisticated CAM software, we can produce real world tool paths and evaluate the final surfaces produced by using combinations of these tools. In Chapter 4 we present a system that can support libraries of multiple tool types, and analyse heuristics used in related research to see if they apply to libraries with multiple tool types. In Chapter 5, we present an optimisation system that performs, for the first time, Tool Sequence Optimisation on a library containing the three main tool types.

There has been no multi-objective approach taken to Tool Sequence Optimisation. There are many benefits in using these techniques. For example, in Machining Parameter Optimisation work, many researchers have used machining time and manufacturing costs as separate objectives. This allows the decision maker to be presented with a set of solutions offering trade-offs between these two objectives, which can provide very useful information. In Chapter 7, we present a multi-objective system that performs Tool Sequence Optimisation, offering trade-off solutions between total machining time and surface tolerance. There is a particular concentration on returning the fastest sequence that can achieve a desired minimum surface tolerance by using preferential search. There has also been very little work done in combining Tool Sequence Optimisation and Machining Parameter Optimisation. In Chapter 8 we present the first multi-objective system that combines the two, returning a selection of solutions that offer a trade-off between total machining time and total tooling costs.

Using sophisticated CAM software enables us to support multiple tool types and produce real world tool paths but it comes at a high computational cost. In (Wang et al., 2005b) and (Wang et al., 2006) parallelisation is employed to speed up optimisation time, using a 37-processor system. In the final experimental work presented in this thesis in Chapter 9, we use a 40-processor parallel architecture that would be easily and inexpensively implementable on an existing office network or commercially available cloud computing system, to speed up the runtime of algorithms on a multi-objective Tool Selection Problem.

2.5 Summary

In this chapter we have presented background information on machining and Evolutionary Computation. We have also reviewed the literature related to Tool Sequence Optimisation and Machining Parameter Optimisation. Analysing the literature, we have identified a number of under-represented areas, particularly the lack of support of tool libraries containing multiple tool types in Tool Sequence Optimisation, the small number of systems combining Tool Sequence Optimisation and Machining Parameter Optimisation and the absence of any work in multi-objective Tool Sequence Optimisation. The experimental work in this thesis will address these issues and add to this body of literature. In the next chapter we will present a formal description of the Tool Selection Problem, methods for evaluating tool sequences and a distributed processing system that allows for thousands of tool sequences to be generated.

Chapter 3

Methods for Evaluating Tool Sequences in Rough Machining

3.1 Introduction

In this chapter, the Tool Selection Problem that is explored throughout this thesis is defined and the experimental setup and evaluation methods are described. The reader is invited to refer to this chapter when reading the experimental work presented in chapters 4, 5, 7, 8 and 9. In addition, a parallel processing system is presented, that provides for the distribution of the computationally expensive tool path and stock model generation across several machines. A version of this system has been implemented, which was used in this thesis to cache whole search spaces. This aided the testing of heuristics and creation of problem specific search parameters in later chapters.

3.2 The Tool Selection Problem

The Tool Selection Problem in rough machining can be defined in different ways, depending on the optimisation objectives. For example in (Ahmad et al., 2010) it is defined in terms of minimising total manufacturing costs. In this thesis, we take a similar tack to (Lim et al., 2001) and (Chen and Fu, 2011), defining it as the minimisation of total machining time. For a given component p, a desired surface tolerance c and a set of tools, $\boldsymbol{S}$, we wish to find the sequence of tools, $\boldsymbol{E} \subseteq \boldsymbol{S}$, where the total machining time of the sequence, $f_m(\boldsymbol{E}, p)$, is minimised and the total thickness of stock from our desired shape, $f_{thickness}(\boldsymbol{E}, p)$ is $< c$. Therefore we wish to minimise:

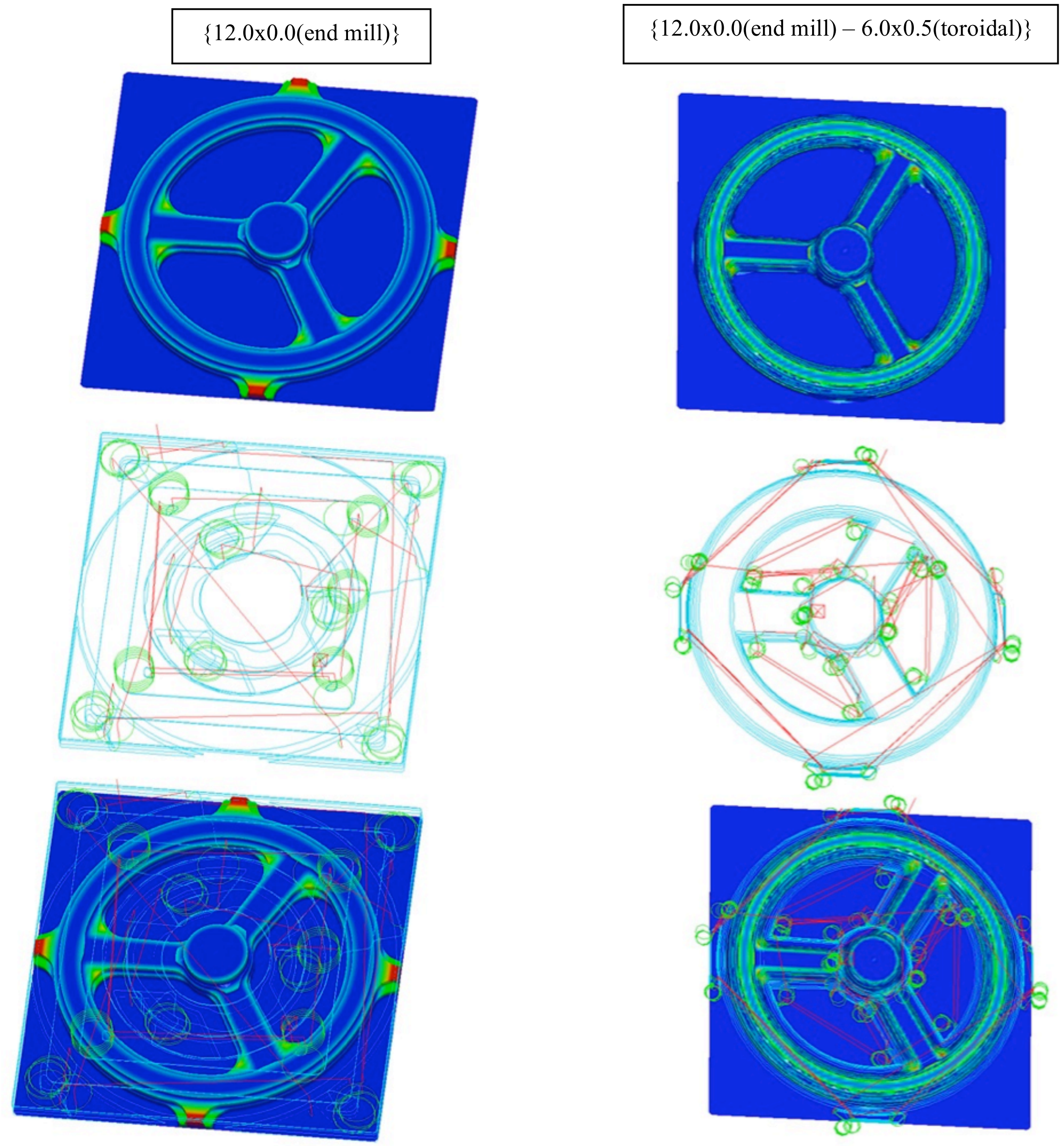
$$f_m(\boldsymbol{E}, p), \boldsymbol{E} \subseteq S, \text{ s.t. } f_{thickness}(\boldsymbol{E}, p) < c \qquad (3.1)$$

In this thesis, the *Tool Selection Problem* and *Tool Sequence Optimisation* are used interchangeably.

3.3 Total Machining Time

To calculate the total machining time for a given sequence of tools, $\boldsymbol{E} \subseteq \boldsymbol{S}$, a tool path must be generated for each tool in the sequence. These tool paths are followed by the cutter and determine where a tool removes material from the workpiece, as well as how the tool moves between points on the workpiece. In the experiments below, tool paths are generated by the CAM software *Machining Strategist* (Vero Software, 2012). This software produces a tool path that consists of four moves. These are the main cutting move, used to remove the desired material; lead-in and lead-out moves, which are used to enter and exit the workpiece smoothly; and rapid moves, which are used to move between different positions on the workpiece without the cutter being in contact with any material. The table feed defines the speed of the main cutting move. In this work, using industrial guidance from Vero Software, the speeds for the other moves are calculated in the following manner:

Figure 3.1. Remaining stock model, tool path and tool path superimposed on the remaining stock model for the best tool combination found, for (a) a 12mm end mill tool and (b) a 6mm toroidal tool following the 12mm end mill tool. On the stock model, red shows high remaining stock, while green shows low remaining stock. On the tool path diagrams, green lines show lead-in and out moves, blue lines show cutting moves, and red lines show rapid moves.



{12.0x0.0(end mill)}

{12.0x0.0(end mill) – 6.0x0.5(toroidal)}

Lead-in move speed = cutting move speed x 0.5

Lead-out move speed = cutting move speed x 0.5

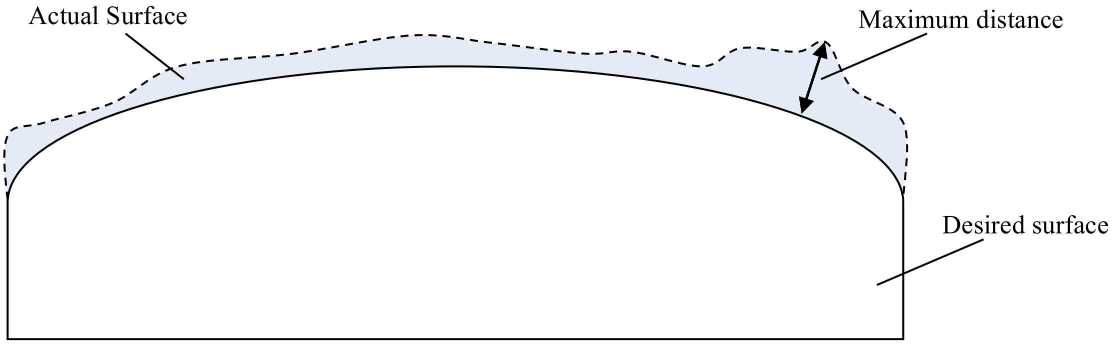
Rapid move speed = 10,000 mm/s

For a given tool sequence, ***E***, consisting of *n* tools and a part, *p*, the total machining time is calculated using,

$$f_m(\boldsymbol{E},p) \;=\; \sum_t^n (C_t\,.\,f_t) + \;(Li_t\,.\,0.5f_t) + \;(Lo_t\,.\,0.5f_t) + \;(R_t\,.\,10{,}000) + T_{ch} \qquad (3.2)$$

where for each tool, *t*, C_t is the cutting tool path length in mm, Li_t is the lead-in tool path length (mm), Lo_t is the lead-out tool path length (mm), R_t is the rapid tool path length (mm) and f_t is the tool's table feed (mm). T_{ch} is a

Figure 3.2. Showing in two dimensions, the point of maximum difference between the actual and desired surfaces produced by a tool sequence.



tool change time, set at a constant value of 30 seconds per tool. Figure 3.1 shows the tool paths for the different moves generated for a 12mm end mill tool and a two tool sequence, a 12mm end mill followed by a 6mm toroidal tool.

3.4 Total Thickness of Excess Material

The goal of machining is to shape a starting blank so that it closely resembles a desired final geometry, to within a certain surface tolerance. Surface tolerance is therefore an extremely important constraint in the optimisation of tool sequences. In the experiments in this thesis, surface tolerance is considered to be the maximum Euclidian distance between a point of excess material on the actual surface and the closest point on the desired surface. A two dimensional example of this can be seen in Figure 3.2. In Figure 3.1, excess material (also called *stock*) is shown in shades ranging from green to red, with red showing the largest distance. The calculation of this maximum distance is performed as a bespoke function added by Vero Software to Machining Strategist and takes advantage of existing data structures for efficiency.

3.5 Roughing Method

As described in the previous chapter, two different methods have generally been used for roughing: (1) layer by layer and (2) remove all accessible material. The second method is used in the experiments in the following chapters, and is the preferred method in industry (Steve Youngs (Vero Software), Personal Communication), due to a lower risk of tool breakages. This method has also been used in related research, such as in (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010; Chen and Fu, 2011).

3.6 Tools

In the literature, discussed in the previous chapter, the majority of Tool Sequence Optimisation work has concentrated on tool libraries consisting of only flat end mill tools. As an industrially important extension, we investigate the use of these tools and additionally ball nose and toroidal tools, the latter of which has not been included in any previous research, to the best of the author's knowledge.

A tool (cutter) is assumed to have two main properties, diameter and corner radius. The flat diameter of the tip of the cutter is given by the simple formula:

Flat diameter = diameter – (2 x corner radius)

The end mill cutter has a corner radius of 0, which means that the tip is completely flat. A ball nosed cutter is defined by having a corner radius of half the cutter diameter, giving a fine tip at the end (flat diameter = 0mm). A toroidal tool does not have a fixed ratio but the corner radius is always less than half the diameter, leaving a flat diameter of more than 0. Each type of cutter has its own advantages, producing different surfaces and operating with different material removal rates.

Throughout this thesis, tools will be denoted in this format, with all units in millimetres (mm):

diameter x corner radius (tool type)

So for example, an end mill tool with a diameter of 12.0mm and a corner radius of 0.0mm will be denoted as:

12.0x0.0(end mill)

Tool sequences will be divided using a '–' symbol. A two-tool sequence consisting of a **12.0x0.0(end mill)** followed by an **8.0x1.0(toroidal)** will be denoted as:

{12.0x0.0(end mill) – 8.0x1.0(toroidal)}

3.7 Parts

Experiments in Chapters 4, 5, 7, 8 and 9 are performed using several real world parts that were provided by our industrial partner, Vero Software. These parts are referred throughout this thesis as Part *x*, where *x* is the index of the part as denoted below. In Table 3.1 and Table 3.2 below, an image of the 3d model of each part is shown, as well as the dimensions of the starting blank that will be reduced to the desired geometry through the machining process. Additionally, we state how long it takes to generate a tool path and stock model using the tool sequence,

{20.0x0.0(end mill) – 12.0x2.0(toroidal) – 6.0x3.0(ball nose)}

on Machining Strategist (Vero Software, 2012) and an Intel Core i5 @ 2.5ghz processor (3210M).

The parts were chosen as canonical examples of real world die moulds that would require tool selection and machining parameter optimisation by a trained professional before manufacture. All of the parts contain curved and sculptured as well as flat surfaces, making it necessary to employ a combination of tool types. Part 1 and Part 3 are smaller than the others, which means that given the right parameters, machining can be performed using short sequences without replacing any tools. This is explored in the tool life experiments in Chapter 8. Parts 2 and 4 are much larger, which requires different strategies to be performed, with interesting trade-offs between tooling costs and machining times. Part 5 is relatively large, with a highly sculptured surface, which is used to evaluate the ability of the optimisation algorithms to deal with complex surfaces. Moving away from purely geometrical and engineering concerns, in Chapters 5 and 7, a hybrid search space created by switching between Part 2 and Part 4 is used to test the algorithms ability to search in rugged and deceptive search spaces, to evaluate their robustness. This artificial component is referred to as Part 2-4-H.

Table 3.1. Showing the 3-dimensional model, the x, y and z dimensions of the starting blank in mm and the processing time (proc. time) in seconds, to generate tool paths and stock models for the 3-tool sequence, {20.0x0.0(end mill) – 12.0x2.0(toroidal) – 6.0x3.0(ball nose)}, for Part 1, Part 2 and Part 3.

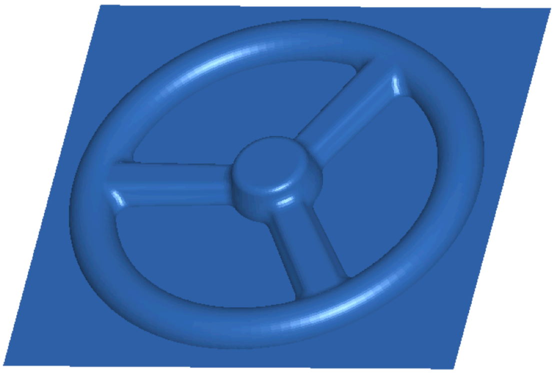
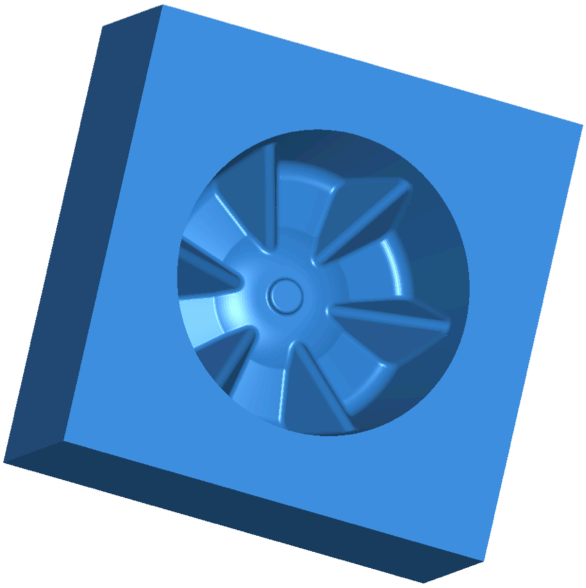
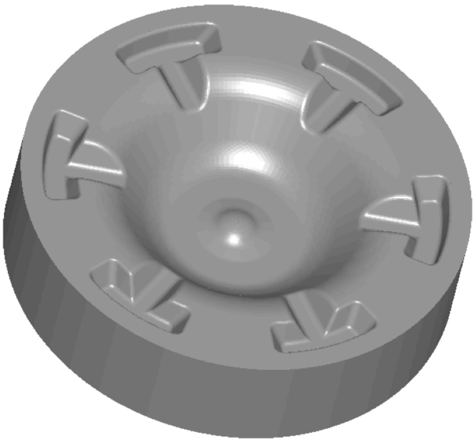
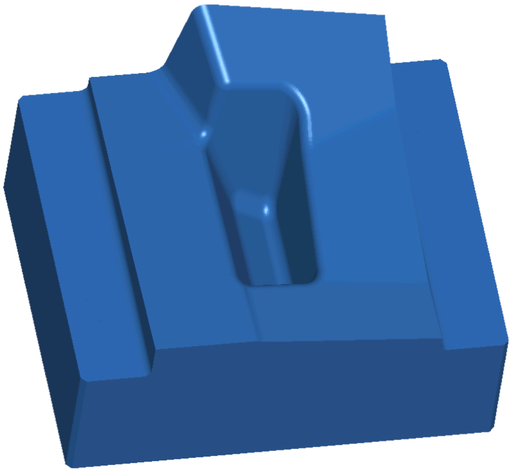
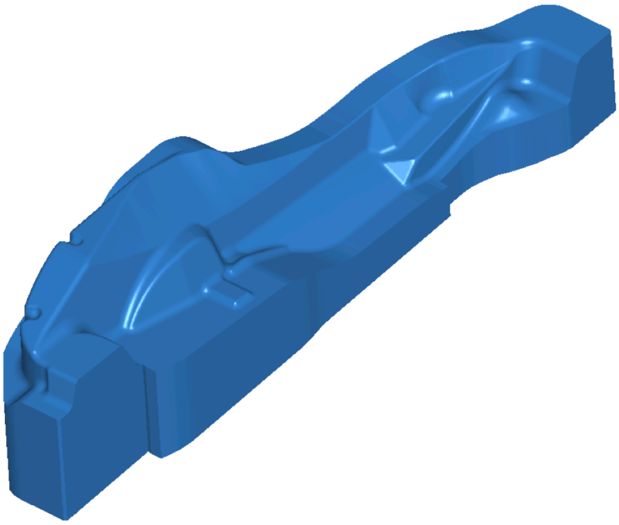
Component Name	3-Dimensional Model	Dimensions of starting blank			Proc. time
		X	Y	Z	
Part 1		100	100	8	35
Part 2		260	260	110	342
Part 3		170	170	42	73

Table 3.2. Showing the 3-dimensional model, the x, y and z dimensions of the starting blank in mm and the processing time (proc. time) in seconds, to generate tool paths and stock models for the 3-tool sequence, {20.0x0.0(end mill) – 12.0x2.0(toroidal) – 6.0x3.0(ball nose)}, for Part 4 and Part 5.

Component Name	3-Dimensional Model	Dimensions of starting blank			Proc. time
		X	Y	Z	
Part 4		240	195	151	224
Part 5		529	124	130	281

3.8 Simulations and Distributed Processing

Industrial CAM software is used to produce realistic tool paths, the same paths that would be used on a real world milling machine, and generates the final geometrical surface of a machining process. Work such as (Lim et al., 2001) has been criticised, for example in (D'Souza et al., 2004), for not using real tool paths that are needed to give an accurate calculation for total machining time. By using Computer Aided Manufacturing (CAM) software, we are able to produce the exact same tool paths that are used in the real machining job, including lead-in and lead-out moves that are an important requirement in real life (Smid, 1999) but have been neglected by most work seen in the literature. The industrial CAM software also allows the freedom to use many different types of tool. Related work that supports tools other than flat end mills has used simulation to assess final geometry (Krimpenis and Vosniakos, 2009; Spanoudakis et al., 2008).

A challenge for using CAM software for evaluating tool sequences is that individual evaluations are time consuming. A main focus for the optimisation work presented in this thesis will be on producing good quality solutions while working within a tight budget of evaluations. Perhaps due to the computational expense of the exercise, even under a constrained environment, no work seen in the literature has performed an exhaustive search of tool combinations for a given part. Providing an exhaustive search of tool combinations allows for an analysis of the search space, which can show the complexity of the task and also helps to evaluate heuristics. It is also useful to be able to rank solutions, in order to evaluate results in a meaningful way. For instance, knowing what the true optimum result is makes it easier to judge the success of different varieties of search procedure.

Caching the search space also allows faster experiments to be performed and supports a large number of repeated trials, which is desirable for designing and testing stochastic search algorithms. Each simulation is expensive in terms of time. A five-tool sequence on a large part can take as long as an hour on an Intel Core i7 running at a clock speed of 4 x 1.66ghz (although this could be less than a minute on a smaller part). By caching all of the results, a search process that can require at least 500 evaluations can be run in seconds rather than days.

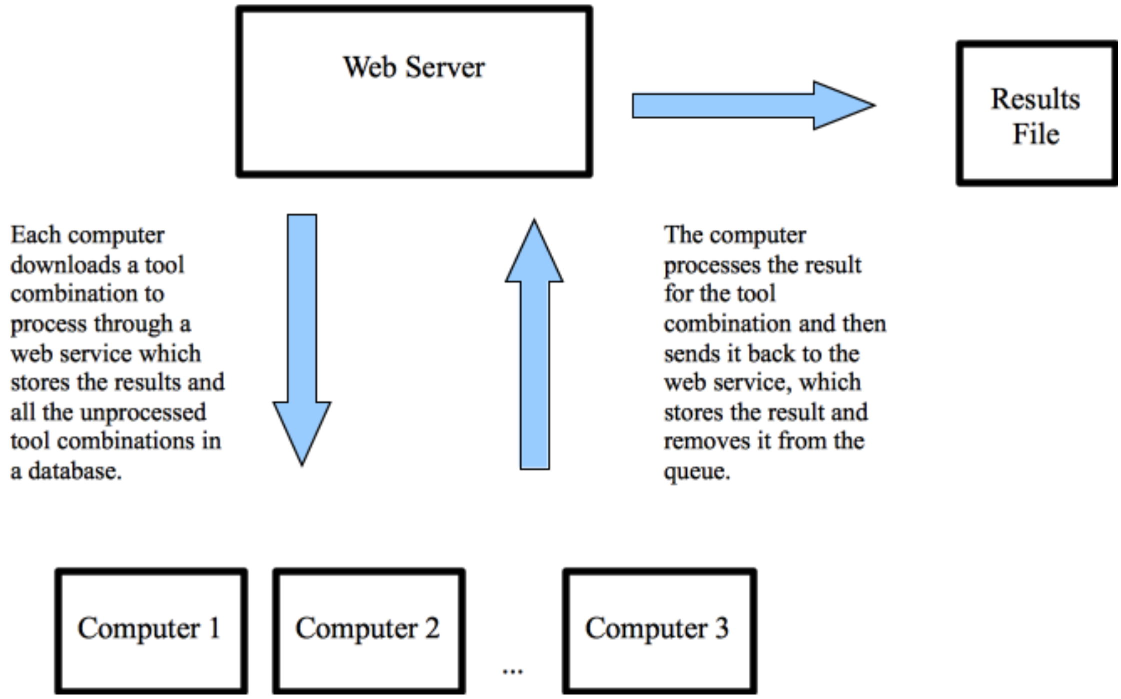
As processing tool sequences can take a long time, we have developed a framework to distribute the task among a network of independent computers. The architecture of this network can be seen in Figure 3.3. A central Internet-hosted service stores in a database all of the legal tool combinations for a given component and tool library. An individual client with a copy of the simulation software will request tool combinations to process from this service. The web application will find a certain number of tool combinations that have not been simulated yet and pass them to the client, which will simulate the combinations in the list and send the results back to the web server. A copy of all of the finished results can be downloaded from the main server at any time and used with a search algorithm or for analysis.

To optimise the processing, we cache fragments of a tool sequence that have been previously calculated either on an individual computer or group of computers. Given the following two tool sequences,

Plan 1: Tool A, Tool B, Tool C

Plan 2: Tool A, Tool B, Tool D

Figure 3.3. The architecture of the distributed processing network used to calculate tool paths and stock models.


Web Server
Results File
Each computer downloads a tool combination to process through a web service which stores the results and all the unprocessed tool combinations in a database.
The computer processes the result for the tool combination and then sends it back to the web service, which stores the result and removes it from the queue.
Computer 1
Computer 2
...
Computer 3

we can see that the first two tools in the sequence are identical for both tool plans. If, in processing Plan 1, we store the state of the simulation after the second tool has been used, for Plan 2 we can skip straight to the last tool, and therefore prevent the need for two full simulations. There are different ways that this can be implemented. Cached solutions can be stored on a single computer's local hard disk. In this case there will be no access to simulations processed by other computers. A second way caching can be implemented is on a local network of computers. Here, the computers can have access to a shared drive, where the cached simulations can be stored. The computers will still be operating independently of any other machines outside the group. A final option for caching is to store the simulations on the global web server and pass the data to machines when it is needed. In this way any machine has access to all previous results and unnecessary reprocessing can be avoided. At the time of writing, many commercial companies, for example Amazon, Google and Microsoft, offer short-term access to a large number of processors, which means that a workshop could inexpensively take advantage of large scale distributed computing without needing to have their own hardware.

In order for network-based caching to work well, it needs to run on a server with the resources available to handle the transfer of a large volume of data, and for client machines to have access to an Internet or local connection with a sufficient amount of bandwidth. The communications overhead can counteract the effect of saving processing time, although this can be mitigated by distributing a certain number of solutions to be processed in batch by each independent machine.

The distributed system introduced above is a small but important contribution to this thesis. It supports the efficient (through sharing cached tool fragments) processing of tool sequences and is implementable on inexpensive, flexible and scalable commercially available distributed computing systems. There are several uses for this system. For the purposes of optimisation, a large number of tool sequences can be evaluated and used for tuning and testing search algorithms. The search space of a tool selection problem can be explored in detail, which can yield greater understanding of the problem, data for building value approximation functions and information on generating heuristics. Finally, this distributed network can be used for testing CAM software, which has already proven to be beneficial to our industrial partner, Vero Software. In addition, this system is used, with only slight modification, to run a single search algorithm in parallel, distributed among many computers, in the work presented in Chapter 9.

3.9 Summary

In this chapter, the Tool Selection Problem tackled throughout this thesis has been introduced and formalised, and the tools, components and roughing style used in later experiments have been described. Methods for evaluating solutions have also been detailed. This chapter should provide enough background for any reader to understand the machining problems explored in this thesis. In addition, a system for distributing the generation of tool paths and stock models in an efficient parallel way has been developed. This system is used in the next chapter to analyse the search space of a particular part in detail and evaluate whether existing heuristics can be used when supporting a wider range of tool types.

Chapter 4

Tool Sequence Optimisation in Rough Machining

4.1 Introduction

In this chapter, a distributed processing system is used to produce tool paths and stock models for every valid tool sequence on a constrained but realistic Tool Selection Problem. An exhaustive search is performed over all feasible solutions, and interesting tool sequences are discussed. In previous research into Tool Sequence Optimisation, such as (Lim et al., 2001; Ahmad et al., 2010), many heuristics have been used. In this thesis, we are extending previous work by using an expanded set of tool types. It is important to evaluate whether existing heuristics are still valid for this changed environment. No related research reported in the literature has performed a detailed survey of the search space in a Tool Selection Problem.

This chapter is organised in the following way. We will first discuss the rough machining experiment that we will perform, and the heuristics that will be applied. We then present the results from an exhaustive exploration of the search space. Analysing the results from this exhaustive search sheds some light on the validity of underlying assumptions made in previous research and the make up of optimal tool combinations. Parts of this chapter appear in (Churchill et al., 2012).

4.2 Roughing Experiment

An exhaustive search of a constrained Tool Selection Problem is presented below. This was performed to provide insight into search space of the tool selection problem. Before embarking on optimisation, it is important to assess prior work and heuristics. An aim of the work in this thesis is to be able to automate the generation of tool sequences, which can make use of end mill, ball nose and toroidal tools. Apart from work found in (Wang et al, 2005a; Spanoudakis et al., 2008), which use ball nose tools, all other researchers have restricted tool libraries to end mills only, and none have used toroidal cutters. Previous heuristics that have been used with end mill-only libraries may not prove effective when there are interactions between different types of tools.

The aim of the search process is to find a tool sequence that will remove material, also known as stock, so that there is less than 1mm deviation from the desired shape, in as short an amount of time as possible. A library of 18 tools is available to create tool sequence plans to rough a sculptured surface, Part 1 presented in the previous chapter (in Section 3.7). The tool library contains six of each of the three tool types discussed above, and their properties are shown in Table 4.1. These tools were provided by our industrial partner, Vero Software, and selected from a tooling manufacturer's catalogue (ITC Ltd., 2012). The number of tools in a tool sequence, referred to as the plan length, is limited to being between 1 and 5. This provides for realistic and useful tool sequences, while creating a

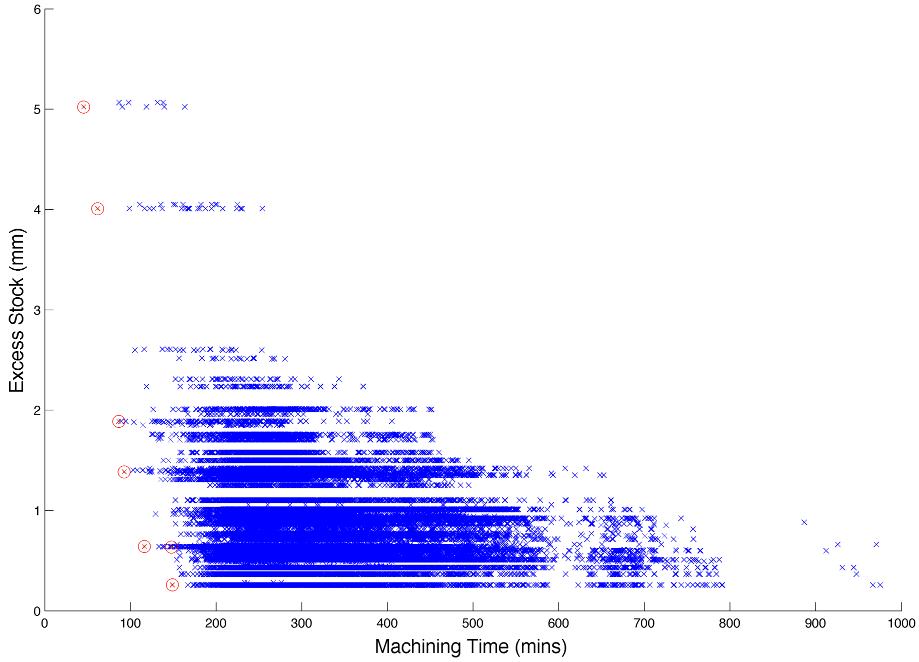
Table 4.1. The attributes of the tools contained in the tool library, provided by Vero Software and taken from (ITC Ltd., 2012).

Tool Type	Cutting Diameter (mm)	Corner Radius (mm)	Stepdown (mm)	Table Feed (mm/min)
End mill	6.0	0.0	1.5	223
End mill	8.0	0.0	2.0	120
End mill	10.0	0.0	2.5	105
End mill	12.0	0.0	3.0	186
End mill	16.0	0.0	4.0	104
End mill	20.0	0.0	5.0	111
Ball nose	6.0	3.0	0.6	200
Ball nose	8.0	4.0	0.8	135
Ball nose	10.0	5.0	1.0	115
Ball nose	12.0	6.0	1.2	105
Ball nose	16.0	8.0	1.6	123
Ball nose	20.0	10.0	2.0	123
Toroidal	6.0	0.5	0.5	200
Toroidal	8.0	1.0	1.0	108
Toroidal	10.0	1.0	1.0	95
Toroidal	12.0	2.0	2.0	167
Toroidal	16.0	2.0	2.0	93
Toroidal	20.0	3.0	3.0	100

search space small enough to be exhaustively evaluated. Only the selection of the tools in the sequence and the plan length is optimised. In these experiments there is no consideration of tool wear, which is a very important factor in industry (and will be implemented in the experiments in Chapter 8). There is also no parameter optimisation, i.e. changing feed, depth of cut or cutting speed, and so here the manufacturer's recommendations are used.

Without any restrictions on the tool sequences in terms of repetition and order, there are around 2 million possible plans. However, many of these are redundant. For example, there is no point in repeating the same tool in a sequence because it removes all the material that it can the first time around. Similarly, if a flat end mill cutter follows another end mill that has a smaller diameter there will be no stock remaining that it is able to remove. The following rules, inspired by (D'Souza et al., 2004), were applied to the tool plans in order to remove this redundancy:

Figure 4.1. Solutions for all tool sequences tested on Part 1 and with the tool library shown in **Table 4.1**. The total excess stock remaining is on the y-axis and total machining time in minutes is on the x-axis. Circled solutions are Pareto optimal.

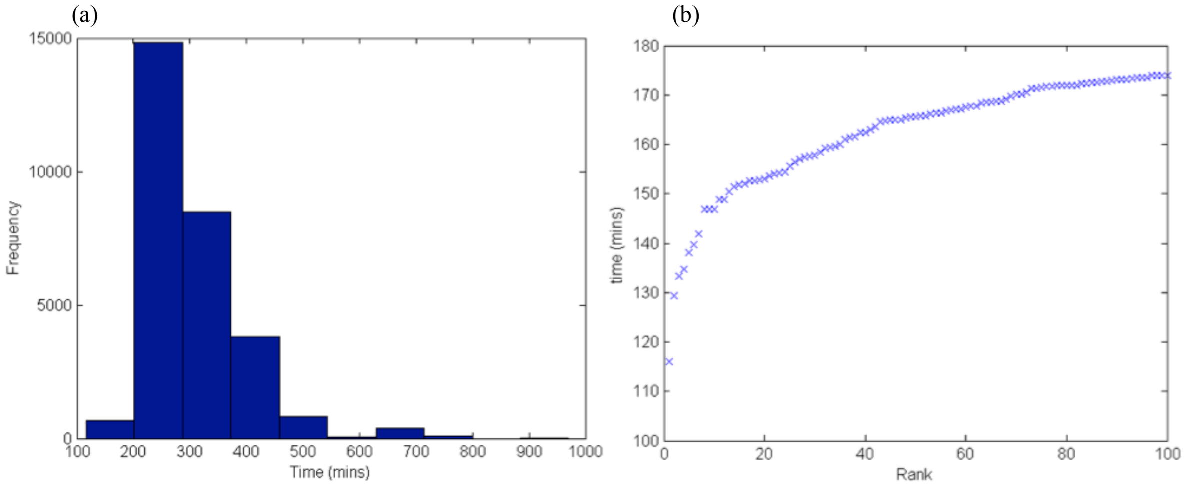


1. Never use a tool that has already occurred in the sequence
2. Never use an end mill cutter whose diameter is ≥ flat diameter of any cutter already used
3. Never use a ball nosed cutter whose diameter is ≥ diameter of any ball nosed cutter already used
4. Never use a toroidal cutter whose diameter is ≥ diameter of any ball nosed cutter already used
5. Never use a toroidal cutter whose flat diameter ≥ diameter of any end mill cutter already used

Versions of these rules have been used in (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010; and Fu, 2011) and prevent tools from appearing in a sequence that can perform no useful work. If for example a larger end mill tool followed a smaller end mill tool in a sequence, the CAM software would not produce any tool paths for the larger tool. These rules have been expanded to include ball nose and toroidal cutters. While no tool paths will be produced for a larger ball nose tool if it follows a smaller ball nose tool, it is possible for useful work to be performed by a larger-diametered ball nose tool following a smaller-diametered tool as there may be sections of the component accessible to the curved edge of the ball nose tool.

Using the above rules, for the tool library in Table 4.1, the total number of sequences was reduced to 39,607. Tool paths and stock models were generated for every one of these legal tool sequences applied to machining Part 1. This was performed using the distributed processing system described in the previous chapter (Section 3.8).

Figure 4.2. Showing for Part 1 and the tool library shown in **Table 4.1**, an exhaustive search of tool sequences gives (a) the distribution of the machining times of solutions with less than 1mm of excess stock and (b) the machining times of the fastest 100 solutions (in rank order) with less than 1mm of excess stock.



4.3 Results

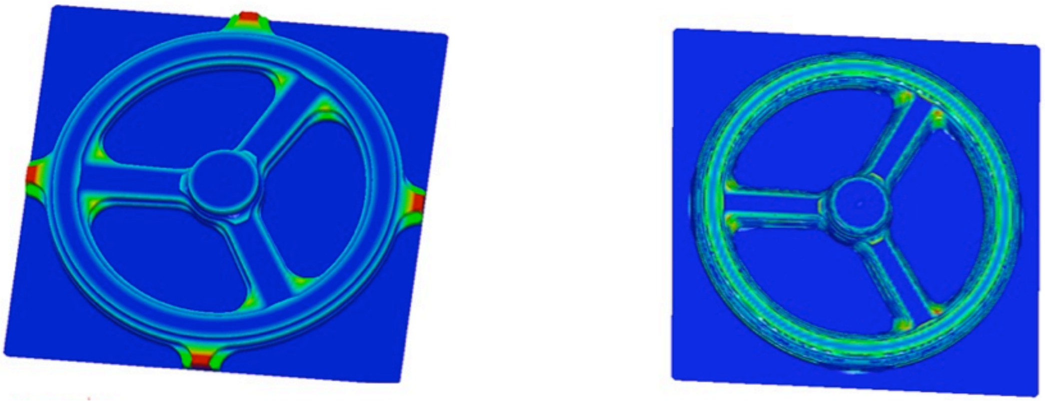
The results for every legal tool combination are shown in the scatter plot in Figure 4.1. The circled solutions are Pareto optimal, meaning that there are no faster solutions with lower excess stock. It is interesting to view solutions in this way because it shows the clear trade-off between machining time and excess material. This suggests that the problem can be framed as a bi-objective problem, which will be investigated further in Chapters 7 and 8.

In Figure 4.1, the solutions appear located in horizontal bands. This is because the excess stock is linked to a particular tool or tool combination in a sequence. There are around 29,000 solutions that achieve a surface tolerance under 1mm. The histogram in Figure 4.2(a) suggests that the search space contains many reasonable solutions but only a small number of very good ones. The majority of solutions lie in the region of 200 – 300 minutes. Figure 4.2(b) shows the machining times of the top 100 solutions. Here we see that there is a relatively large gap between the optimal solution and the second best but the other solutions increase in machining time in a fairly smooth manner. While finding the optimal solution is desirable, in a real world situation producing a good solution to the problem in a reasonable amount of time is often acceptable, given the computational expense of optimisation in terms of producing stock models and tool paths. Figure 4.2(b) shows that with this part and tool library, there could be many acceptable solutions.

4.3.1 Fixing the Last Tool in the Sequence

Analysing the results for all of the tool sequence combinations for this part reveals many interesting features. The best plan is {12.0x0.0(endmill) – 6.0x0.5(toroidal)} with a score of 116 minutes. Figure 4.3 shows the remaining stock after each individual tool has been used in the sequence. The second best plan is {12.0x0.0(endmill) – 8.0x1.0(toroidal)}, which takes 129 minutes. Using a smaller tool – the 6mm diameter toroidal cutter rather than the 8mm toroidal – at the end of a sequence leads to a more efficient tool plan. This is interesting because using the heuristics in (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010), the {12.0x0.0(endmill) – 6.0x0.5(toroidal)} solution could not be found.

Figure 4.3. Remaining stock model, tool path and tool path superimposed on the remaining stock model for the best tool combination found, {12.0x0.0(endmill) – 6.0x0.5(toroidal)}. On the stock model, red shows high stock, while green shows low excess stock. On the left we see the remaining stock after using the 12.0x0.0(endmill) tool and on the right, after the 6.0x0.5(toroidal) tool that follows.



12.0x0.0(endmill)

12.0x0.0(endmill) – 6.0x0.5(toroidal)

Table 4.4. Tool path lengths and times taken for the different types of cut when 12.0x0.0(end mill) is the first tool in the sequence.

	Cutting	Rapid	Lead-In	Lead-Out
Path Length (mm)	6578.30	1190.13	2356.33	88.88
Time taken (mins)	35.37	0.12	25	0.96

Table 4.4 Tool path lengths and times taken for the different types of cut when 6.0x0.5(toroidal) follows 12.0x0.0(end mill). This is the best solution found.

	Cutting	Rapid	Lead-In	Lead-Out
Path Length (mm)	6978.11	2483.41	1615.79	257.8
Time taken (mins)	34.89	0.25	16.16	2.58

Table 4.4. Tool path lengths and times taken for the different types of cut when 8.0x1.0(toroidal) follows 12.0x0.0(end mill). This is the second best solution found.

	Cutting	Rapid	Lead-In	Lead-Out
Path Length (mm)	3951.26	1471.3	1510.96	133.06
Time taken (mins)	36.59	0.15	27.98	2.46

This means that employing the heuristic used by these researchers, the system would be incapable of locating the optimal solution.

The tool paths in Tables 4.2 – 4.4 give more details for this result. The larger diameter 8mm toroidal tool has a much shorter cutting path and a similar lead-in. If both tools operated at the same speed, the larger tool's work would be completed much more quickly. However, the 6mm toroidal has a table feed of 200mm/s compared to a much slower 108mm/s for the 8mm. This is exacerbated by lead-ins moves operating at a slower speed, leading the smaller tool to complete its journey approximately 14 minutes sooner.

In (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010), a "critical tool" is fixed at the end of the sequence, before search begins on the rest of the sequence. This tool is chosen by finding the largest single cutter that is able to produce the required surface tolerance. However, here we see that using this heuristic can lead to inefficient results because table feeds are not taken into account. Tools can have very different properties. They can have different numbers of teeth and can be made out of a diverse range of materials, all of which affect the table feed. It may also be the case that relying on a geometrical model without calculating tool paths, as is done in (Lim et al., 2001), will lead to suboptimal sequences. This is because the true machining times of the individual tools are not considered. Simplifying the tool paths could also cause problems because lead-ins and lead-outs, which are necessary for smooth surfaces in real world machining (Smid, 1999), have slower cutting speeds than the normal cutting path. In (D'Souza et al., 2004; Ahmad et al., 2010), machining time is calculated using only cutting and rapid moves, meaning that lead-ins and outs are not included in the machining tool path, even though the tables above show that they can contribute to a considerable part of the total machining time.

4.3.2 Different Sequences with Similar Machining Times

Another interesting observation is made from looking at the third and fourth best solutions,

3. {12.0x0.0(end mill) – 6.0x0.0(end mill) – 6.0x0.5(toroidal)}
4. {20.0x0.0(end mill) – 12.0x2.0(toroidal) – 6.0x0.5(toroidal)}

They take almost the same amount of time, 133 and 135 minutes. As with the best result, both of these plans utilise the 6.0mm toroidal cutter at the end of the sequence. However, an extra tool is used in the sequence, which significantly shortens the length of the last tool's cutting path. This reduces the load on the smallest tool, which a process planner may prefer because it means there will be less stress on the more delicate cutter. Out of the two, the second of these plans, while being slightly slower, would probably be chosen as it uses much larger, and therefore stronger tools. This highlights an issue that will be addressed later in this work, which is that it is desirable for a machinist to have a range of solutions to choose from. Optimisation, especially with heuristics, tends to guide search in a certain direction or use a subset of conditions in a value function. It is impossible to include the needs of every individual user in a general framework but providing several solutions can help decision makers make the most informed choice, based on their particular situation.

4.3.3 The Last Tool in the Sequence Does Not Entirely Define the Final Surface

Another discovery from this extensive search is that there are tool combinations that are able to achieve a lower surface tolerance together than they can individually. The tool combination,

{12.0x0.0(end mill) – 6.0x0.0(end mill) – 10.0x5.0(ball nosed)}

is ranked 18th out of all the combinations, taking 153 minutes. Table 4.5 below shows the surface tolerance the tools can reach individually and together. The lowest maximum distance of excess stock that can be individually achieved is 1.06mm. However, in combination this is reduced to 0.87mm. This is better illuminated in Figure 4.4 below, where we see the stock profiles left by 6.0x0.0(end mill) and 10.0x5.0(ball nose) when used individually as well as together. In (D'Souza et al., 2004), an assumption is used where the profile left by a larger tool is considered to be the geometrical subset of a smaller tool, meaning that a smaller diameter tool can always completely machine the profile left by a larger diameter tool. This is used on libraries containing only end mill cutters, where this assumption holds. However, our result shows that this assumption should not be used if the tool library contains

Figure 4.4. The final surface achieved producing Part 1, when using the tool sequences a) 6.0x0.0(end mill), b) 10.0x5.0(ball nose) and c) 6.0x0.0(end mill) followed by 10.0x5.0(ball nosed). We see that both (a) and (b) individually cannot produce less than 1mm of stock but they can in combination.

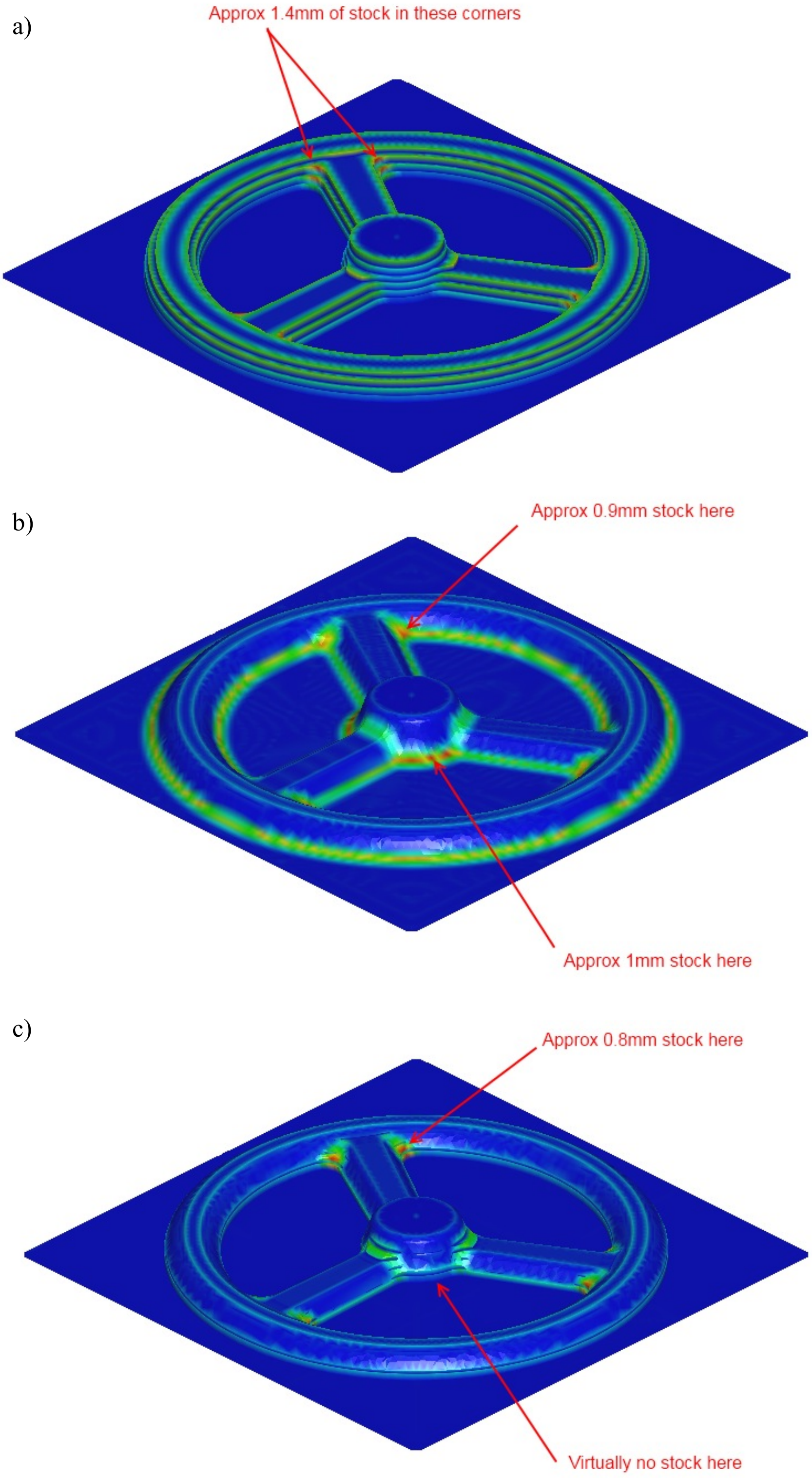


Table 4.5. The surface tolerances achieved by 3 tools individually and together on tool sequence, {12.0x0.0(end mill) – 6.0x0.0(end mill) – 10.0x5.0(ball nosed)}.

Tool	Maximum Surface Tolerance (mm)
12.0x0.0(end mill)	4.01
6.0x0.0(end mill)	1.40
10.0x5.0(ball nosed)	1.06
All three together	0.87

ball nose cutters. It also suggests that fixing the last tool in the sequence before starting the optimisation process, as is done in (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010), may prevent optimal solutions from being found.

4.3.4 Toroidal Tools

A final point of interest is the solution ranked sixth overall,

{12.0x2.0(toroidal) – 6.0x0.5(toroidal)}.

This shows a combination made entirely of toroidal tools. A large proportion of the related research reviewed above concentrates solely on end mill tools but this result shows that a very efficient tool plan can be formed without using a single one. None of the projects surveyed in the literature consider toroidal tools but they can be very useful because they are able to produce both flat and curved surfaces (Kalpakjian and Schmid, 2008).

4.4 Discussion

In this chapter we have, for the first time, carried out a sweep through the entire search space of a Tool Selection Problem. The results have shown this to be important for many reasons. Firstly, it enables the solutions to be ranked, so that we can properly assess the performance of different search algorithms. Secondly, it provides a way to test whether we should consider applying existing heuristics to our approach, which differs from previous work by using three different types of cutter. The search space analysis led us to four important outcomes for our system:

1. Fixing the last tool in the way described in (Lim et al., 2001), (Ahmad et al., 2010) and (D'Souza et al., 2004), can result in suboptimal solutions, as table feed is not taken into account.

2. Combinations of different shaped tools can achieve lower surface tolerances than the individual constituents are able to, meaning that the assumption used in (D'Souza et al., 2004) should not be used with libraries consisting of different tool types.

3. Good solutions can be produced without using any end mill tools. This means that if possible tool libraries should consist of a variety of tool types.

4. Good solutions exist that use different combinations of tools. As a machinist may have many factors to consider based on their individual situation, it is desirable to present multiple solutions to the problem.

4.5 Summary

In this chapter we have presented an in depth analysis of the search space on a single part, using a realistic library of tools consisting of the three main types of cutter. Through this, a number of interesting features have been identified that show that heuristics and assumptions used in related work could prevent optimal tool sequences from being discovered. This enables us to develop our own approach, using fewer heuristics and supporting variable length tool sequences, which we will integrate with metaheuristic algorithms in the next chapter in order to evaluate the effectiveness of different search techniques abilities to find optimal solutions.

Chapter 5

A Metaheuristic Approach to Tool Sequence Optimisation in Rough Machining

5.1 Introduction

In the previous chapter, we have seen that tool sequences display a wide range of behaviour, with some performing far better than others. This suggests that we could benefit greatly from optimisation. We also saw that using multiple cutter types invalidates many heuristics used in existing work on Tool Selection problems in rough machining. Given that we have a combinatorial problem where it is difficult to predict the interactions between tools with different geometrical properties, Evolutionary Computation is a good candidate for optimisation, especially as there is a want for multiple solutions. Once a representation scheme and variation operators have been determined, there is an opportunity to apply multi-objective and parallel Evolutionary Algorithms, which are explored later in this thesis.

In this chapter, a problem-specific representation scheme and novel search operators are introduced, which support variable length tool sequences, flexible ordering and multiple cutter types. A Genetic Algorithm (GA) is tested against Hill Climbing approaches to provide a baseline comparison, identify issues with the evolutionary approach, and provide a method for including local search in the GA. In addition to the GA, we evaluate the performance of three other metaheuristic algorithms on the Tool Selection Problem defined in Chapter 3. These are Stochastic Hill Climbing (SHC), Random Restart Stochastic Hill Climbing (RRSHC), and a hybridisation of the GA and SHC (HGA). The algorithms are assessed on their ability to cope with two competing demands – finding good quality solutions and keeping the number of potentially expensive evaluations low.

This chapter is organised in the following way. First we introduce the experiment and similar work that has been reported in the literature. Next we introduce the methods used in the experiments, starting with the representation scheme and variation operators and moving onto the exact implementations of the search algorithms. A parameter sweep is presented for each algorithm. A first set of experiments is then performed on Part 1 (presented in Section 3.7 in Chapter 3), and compares two exploration (mutation) operators. Using a gradual method that biases exploration towards small changes in sequences is shown to significantly improve both search quality and speed for all algorithms. A second set of experiments is performed on three additional components. One of these has a difficult search space with several local optima and sharp jumps in fitness, created by an error in the simulation process, which combined the search space of Part 2 with that of Part 4 (this will be denoted Part 2-4-H). These errors meant that certain tool sequences were given inaccurate shorter tool paths and lower maximum stock values, distorting the search landscape. This “part” thus provides a useful test for how the algorithms respond to difficult search spaces with several local optima. On three of the four tested parts, RRSHC and HGA performed the best, with the GA performing worst. However, on Part 2-4-H, in the presence of many local optima, RRSHC and the GA outperformed HGA and SHC, with the latter performing particularly badly. Parts of this chapter appear in (Churchill et al., 2012).

5.2 Background

There are several elements to Tool Selection problems that make them difficult to solve. Tools operate at different speeds and the lengths of their tool paths depend on preceding members of the sequence. This means that smaller tools that operate at faster speeds can have advantages over larger, slower tools, depending on the amount of material that needs to be removed. The interaction between tools of different types, i.e. different shaped cutters, can make it difficult to predict what the final geometrical profile of a tool sequence will be. In the previous chapter we also saw that not all existing heuristics apply to tool libraries containing multiple tool types.

In this thesis we mainly concentrate on population-based evolutionary search techniques for tackling Tool Selection and machining problems. This is used because of their robustness in finding good solutions in changing search landscapes, and importantly, their population-based search naturally lends itself to finding multiple solutions in a single search run (Goldberg, 1989; Giagkiozis et al., 2013). In later chapters, the extensibility of Genetic Algorithms will be shown, when the work presented in this chapter is adapted to explicitly provide multiple solutions (in Chapter 7) and multiple objectives (in Chapter 8). Another advantage of population-based search is the easy parallelism offered by the approach, which is explored in Chapter 9.

In this chapter, we investigate the ability of a Genetic Algorithm (GA) and three other metaheuristic algorithms to traverse the tool sequence search space of four parts. The GA is compared to Stochastic Hill Climbing (SHC) and Random Restart Stochastic Hill Climbing (RRSHC) as a baseline, and also sets up a framework for integrating local search into the evolutionary algorithm. A memetic algorithm is created by hybridising the GA with SHC (HGA). On each part, the algorithms have to select tool sequences of between 1 and 5 tools in length, using the same 18-tool library.

As discussed in Chapter 4, previous researchers have employed Evolutionary Computation for tool sequence optimisation, for example in (Vosniakos and Krimpenis, 2002; Spanoudakis et al., 2008; Krimpenis and Vosniakos, 2009; Ahmad et al., 2010; Chen and Fu, 2011). These projects have been successful but have had limitations such as fixing the last tool in the sequence and using only simple tool libraries. In this chapter we present a less restrictive approach with a more industrially relevant tool library, consisting of the three main tool types.

No tool sequence optimisation work in the literature has used Hill Climbing or hybrid evolutionary techniques. However, (Baskar et al., 2006) applied both to machining parameter optimisation on a milling task and found the hybrid to offer considerable improvements. In (Wang et al., 2005b), a parallel Genetic Algorithm hybridised with parallel Simulated Annealing is used to optimise cutting speed and feed rate in a milling application, and again it was found that the hybrid performed better than the individual algorithm. These results suggest that a memetic approach could be useful for Tool Selection problems.

Another difference between the experiments in this chapter and previous studies is the concentration on limiting the number of potentially expensive evaluations. By evaluating using real world Computer Aided Manufacturing (CAM) software, we are able to use complex tool libraries and can include many real world conditions such as tool holders and flute lengths. Real tool paths are generated, meaning that accurate machining times can be calculated, and a 3D stock model is produced which provides both a measure of the size and location of excess material and also of general surface roughness, which can be useful to process planners. A disadvantage of this is the computational cost, which means that evaluations become the limiting factor. There is a clear trade-off between the quality of an obtained solution and the time taken to find it. The investigation in this chapter aims to assess how well different stochastic algorithms perform with regards to these two competing objectives.

Another important evaluation criterion is repeatability. Putting ourselves in the shoes of a machinist, it is important that an optimisation method will produce consistently good results. Unlike similar work in the literature, we test each algorithm over a large number of trials, which allows us to make statistical inferences and properly assess the performance of an algorithm in the best, average and worst case. This approach is made possible by using the distributed processing system introduced in Chapter 3 (Section 3.8).

5.3 Methods

In this section we will first introduce the search operators and evaluation method and then describe in detail the exact way that each algorithm has been implemented to solve the Tool Selection problems presented in the results section. Three of the algorithms, RRSHC, GA and HGA also have a number of parameters and stopping conditions used to control the depth and breadth of the search, which are explored through a parameter sweep presented after the implementation of each algorithm has been described.

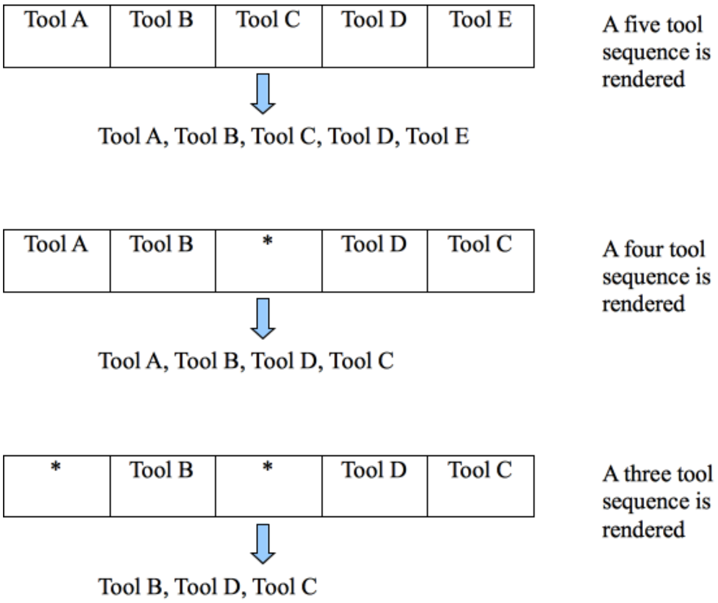
5.3.1 Search Methods

5.3.1.1 Representation

All of the metaheuristic algorithms use the same representation scheme. A tool sequence is stored as a string (genotype) of L elements (chromosomes), where L is the maximum allowed tool sequence length. Each element in the string stores a pointer to either a tool in the tool library or a blank character '*'. The string is read from left to right, determining the order of the tools in the sequence. Examples can be seen in Figure 5.1, which shows the rendering of a three, four and five-tool sequence.

The use of the '*' symbol allows for variable length tool sequences, without enforcing an explicit ordering. This differs from the representation used in (Ahmad et al., 2010; Chen and Fu, 2011). Both of these systems use a binary representation where each element of a binary string corresponds to a tool. The order of the tools in the sequence is determined by diameter sizes, sequenced from largest to smallest. This is not restrictive in their systems because only end mill tools are considered. However, in our system, large diameter ball nosed tools can be placed before or after smaller diameter end mill tools, which would be impossible to realise using a fixed ordered genotype. This supports more

Figure 5.1. Showing three examples of the scheme used to represent tool sequences by the metaheuristic algorithms, consisting of five, four and three tools. The maximum tool sequence length, *L*, is five.


Tool A
Tool B
Tool C
Tool D
Tool E
A five tool sequence is rendered
Tool A, Tool B, Tool C, Tool D, Tool E
Tool A
Tool B
*
Tool D
Tool C
A four tool sequence is rendered
Tool A, Tool B, Tool D, Tool C
*
Tool B
*
Tool D
Tool C
A three tool sequence is rendered
Tool B, Tool D, Tool C

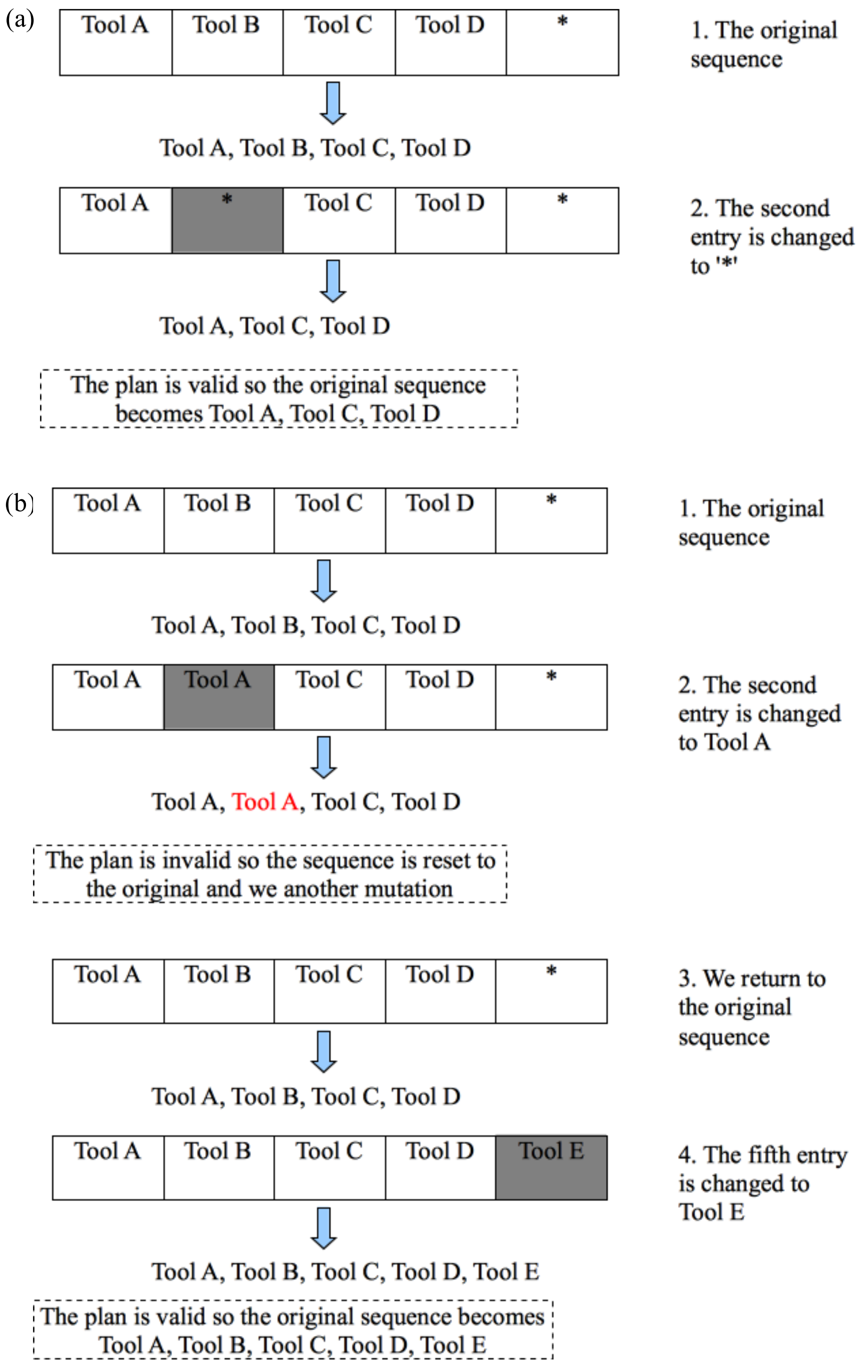
complex interactions between different types of tools, which was shown in the previous chapter to lead to interesting solutions.

5.3.1.2 Global Mutation Operator

The mutation operator is used to explore the search space and move between solutions. Genetic Algorithms use this together with recombination for exploration, while the Hill Climbing methods use mutation alone. Two different mutation operators are tested with the algorithms in this chapter, a "*global*" method described here, and a "*gradual*" method described below.

In the *global* method, mutations are achieved by changing one of the entries in the genotype to either one of the tools in the library or to the '*' symbol, representing an empty entry. An equal weighting is given to all of these options. This process repeats, resetting to the original combination each time, until a legal combination is found, according to the rules described in Section 4.2 in the previous chapter. Every valid tool sequence one mutation away from the original is considered to be a neighbouring sequence. An example of the Global Mutation Operator is illustrated in Figure 5.2(a) showing a valid mutation, and Figure 5.2(b), showing an invalid mutation. Mutating in this way provides an unbiased exploration of the search space and the ability to quickly move to new solutions.

Figure 5.2. An example of (a) a valid mutation and (b) an invalid mutation. Both examples use the global mutation operator. If an invalid mutation occurs the sequence is reset to the original and another mutation attempt is made.


(a)
Tool A
Tool B
Tool C
Tool D
*
1. The original sequence
Tool A, Tool B, Tool C, Tool D
Tool A
*
Tool C
Tool D
*
2. The second entry is changed to '*'
Tool A, Tool C, Tool D
(b)
Tool A
Tool B
Tool C
Tool D
*
1. The original sequence
Tool A, Tool B, Tool C, Tool D
Tool A
Tool A
Tool C
Tool D
*
2. The second entry is changed to Tool A
Tool A, Tool A, Tool C, Tool D
Tool A
Tool B
Tool C
Tool D
*
3. We return to the original sequence
Tool A, Tool B, Tool C, Tool D
Tool A
Tool B
Tool C
Tool D
Tool E
4. The fifth entry is changed to Tool E
Tool A, Tool B, Tool C, Tool D, Tool E

The plan is valid so the original sequence becomes Tool A, Tool C, Tool D

The plan is invalid so the sequence is reset to the original and we another mutation

The plan is valid so the original sequence becomes Tool A, Tool B, Tool C, Tool D, Tool E

5.3.1.3 Gradual Mutation Operator

The Global Mutation operator described above could be quite disruptive for two reasons. Firstly, a tool can be changed to any other tool, as long as the sequence remains valid. This means that a very large tool can be swapped with a small tool, which has a large effect on a sequence. Secondly, there is a much smaller probability of a tool being removed or inserted than being changed because mutations are applied directly to the genotype (sequence string).

A *gradual* mutation operator was designed to avoid these problems. It works in the following way. Firstly, when tools are inserted or changed, the two tools that are most similar in size to the existing tool, in the case of a tool change, or to the neighbours in the sequence, in the case of a tool insert, have twice the probability of being selected than the two tools next most similar in size, which themselves have double the probability of being selected compared to the remaining tools in the library.

Secondly, the new mutation process operates on the phenotype (i.e. the final tool sequence) and gives a 0.3 chance of inserting or removing a tool and a 0.4 chance of changing an existing tool. In this way the mutations are less drastic and the changes more gradual. The Gradual Mutation operator is described in detail below for the case of a tool replacement, tool removal, and tool insertion. In Section 5.4 below, the *global* and *gradual* mutation operators are compared on a Tool Selection task.

5.3.1.3.1 Gradual Tool Change

In the Gradual Mutation operation, new tools have a higher probability of selection if they are of a similar size to the tools they are replacing or inserting between. Firstly, the tools in the library are sorted. End mill and toroidal cutters are ordered on their flat diameters and ball noses are separately ordered by their diameters but considered smaller than the other tool types. The extreme tools are located, meaning the largest and smallest tools that could be placed into the sequence at the desired point without breaking the rules defined in Section 4.2 of Chapter 4. The tools above and below the tool being changed in the ordering are placed into the first group (G_a). The two tools next to these are placed in a second group (G_b) and all other valid tools are placed into a third group (G_c). Tools in G_a have twice the probability of being selected as those in G_b. Tools in G_b have twice the probability of being selected as those in G_c.

The algorithm to find the new tool works in the following way:

1. Choose the position in the sequence where the new tool will be inserted or replace an existing tool, z
2. If it is a replacement operation, identify the tool being replaced, t_{repl}
3. Identify a starting tool, t_{start}, the tool positioned before t_{repl} in the sequence
4. Identify an ending tool, t_{end}, the tool positioned after t_{repl} in the sequence
5. Find the list of tools, L, that are smaller than t_{start} and larger than t_{end}
6. Sort L so that it is in size order, giving L_{sorted}
7. If L < 2, return the single tool, otherwise go to 8
8. While the size of G_a, is < 2, remove the tool closest in size to t_{repl} (for a replacement) or t_{start} (for an insertion) from L_{sorted} and add to G_a
9. If size L > 2, while the size of G_b, is < 2, remove the tool closest in size to t_{repl} (for a replacement) or t_{start} (for an insertion) from L_{sorted} and add to G_b
10. While the size of L_{sorted} > 0, remove the tool closest in size to t_{repl} (for a replacement) or t_{start} (for an insertion) from L_{sorted} and add to G_c
11. Set $w = \frac{1}{Length\ of\ L}$
12. For tools in G_a, assign a probability weighting $4w$

13. For tools in G_b, assign a probability weighting $2w$
14. For tools in G_c, assign a probability weighting w
15. Select a tool biased by the weightings

An example of a tool change can be seen in Figure 5.3.

5.3.1.3.2 Tool Removal

Removing a tool from the sequence is given an explicit probability of 0.3, a decision made from preliminary experiments. A tool from the sequence is chosen at random from a uniform distribution and its entry in the genotype is changed to the '*' symbol. Figure 5.4 below illustrates this process.

5.3.1.3.3 Tool Replacement

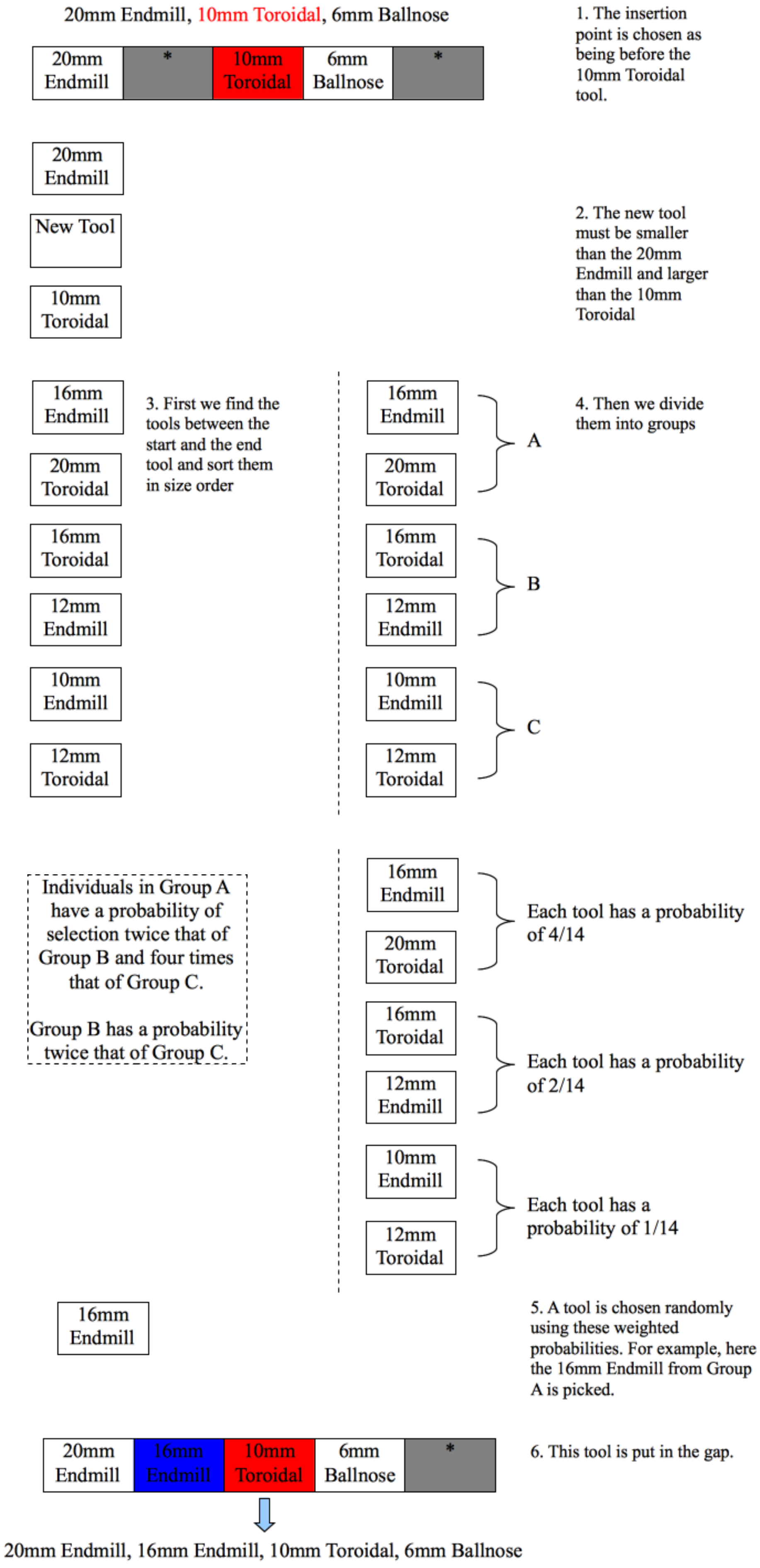
Replacing a single tool in the sequence, without modifying the sequence length, is given a slightly higher probability of 0.4, again determined from preliminary experiments. Unlike insertion, this method requires a simple change to the genotype, which can be seen in Figure 5.5. A tool in the sequence is selected at random from a uniform distribution. This tool is then replaced with a different tool using the gradual tool change method described above.

5.3.1.3.4 Tool Insertion

Inserting a tool into a sequence is a more complicated procedure. Unlike the removal and replacement operations, there are situations where several alterations need to be made to the genotype in order to realise the changes. The first step in the insertion process is to randomly choose a point in the sequence where a new tool is inserted. Three different scenarios are possible – a tool being inserted at the beginning, the middle and the end of a sequence. As it is the simplest, the last case is described first. The position of the last tool in the sequence is identified in the genotype. If there is an empty space in the genotype after the last tool, i.e. a '*' symbol, a new tool is inserted here. In the absence of this, an empty space is identified in the sequence before the current last tool. All the tools in front of this space then have to shift forwards to remove the space present earlier in the sequence and create a gap at the end. Once this gap is available a new tool will be inserted inside it. This process can be seen in Figure 5.6 below.

If a new tool is chosen to be inserted at the beginning of the sequence we apply a similar procedure. We identify the position of the first tool in the genotype. If there is a gap directly before this tool, we insert a new tool here. Otherwise we create a gap by finding the first gap after the first tool and then shifting all of the tools basckwards to create a new gap at the beginning of the sequence, where a new tool is inserted. An example of this can be seen in Figure 5.7.

Figure 5.3. An example of a gradual tool change, when using the Gradual Mutation operator.



1. The insertion point is chosen as being before the 10mm Toroidal tool.

2. The new tool must be smaller than the 20mm Endmill and larger than the 10mm Toroidal

3. First we find the tools between the start and the end tool and sort them in size order

4. Then we divide them into groups

Individuals in Group A have a probability of selection twice that of Group B and four times that of Group C.

Group B has a probability twice that of Group C.

Each tool has a probability of 4/14

Each tool has a probability of 2/14

Each tool has a probability of 1/14

5. A tool is chosen randomly using these weighted probabilities. For example, here the 16mm Endmill from Group A is picked.

6. This tool is put in the gap.

Figure 5.4. An example of a tool removal using the Gradual Mutation operator**.**

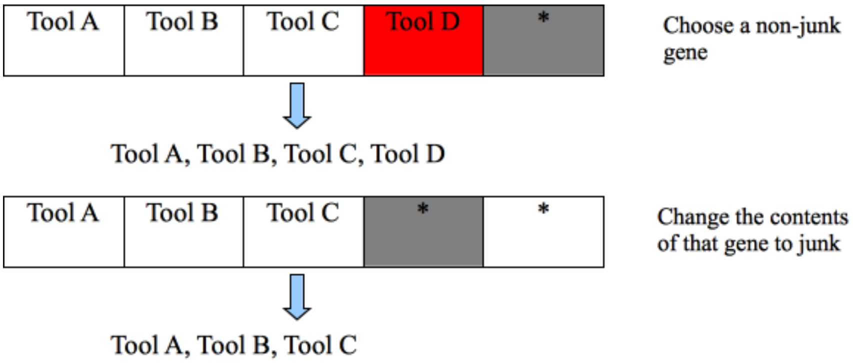
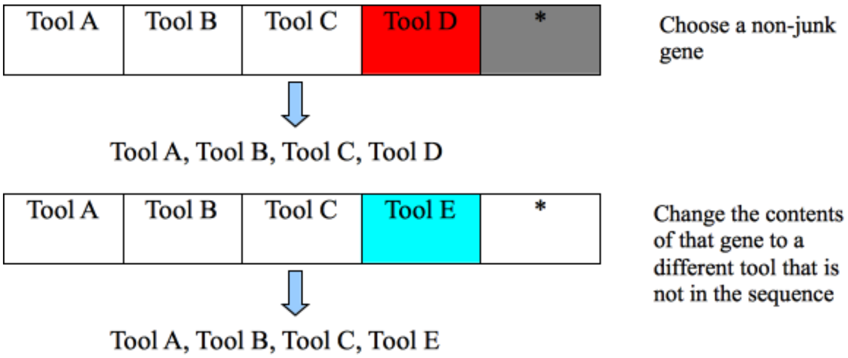
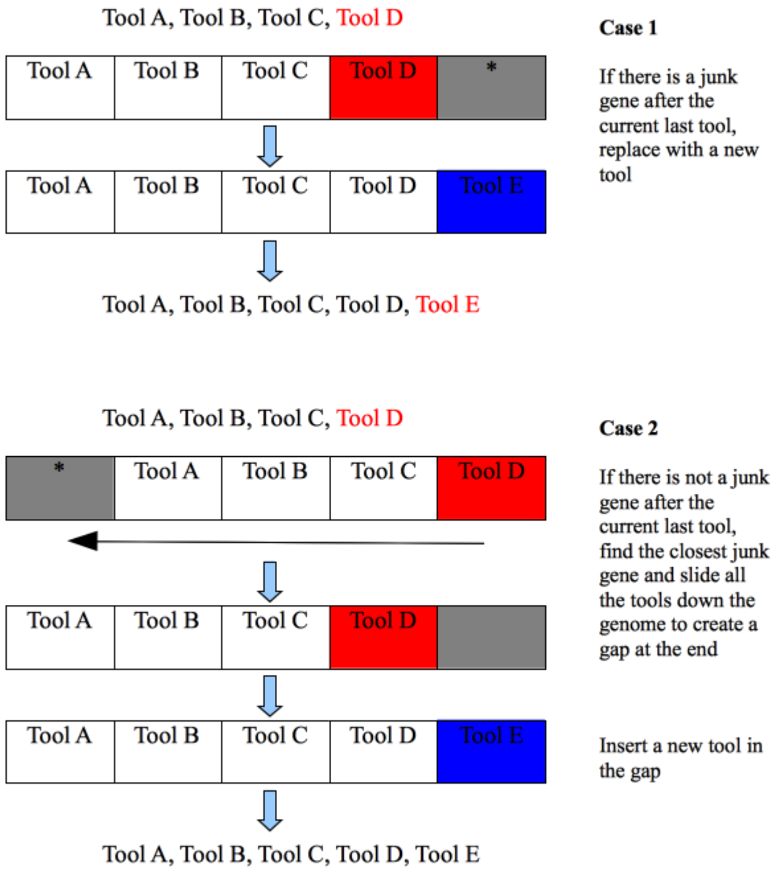


Figure 5.5. An example of a tool replacement using the Gradual Mutation operator**.**


Tool A
Tool B
Tool C
Tool D
*
Choose a non-junk gene
Tool A, Tool B, Tool C, Tool D
Tool A
Tool B
Tool C
Tool E
*
Change the contents of that gene to a different tool that is not in the sequence
Tool A, Tool B, Tool C, Tool E

In the final case, a tool is inserted at a point in the middle of the sequence. If there is not a gap in the genotype directly before the chosen tool, the situation is more complicated. First the element in the genotype directly after the chosen tool is checked to see if there is a gap. If there is then the chosen tool is shifted upwards to fill this gap and a new tool is inserted in the chosen tool's position. If not, there is a check to see if there is a gap before the chosen tool. If there is, all the tools before the tool in question are shifted backwards to create a gap before the chosen tool. If this is also not the case, then a gap is found after the insertion point. The tool at the insertion point and all others up to the gap are shifted forward to create a gap before the insertion point, where a new tool is inserted. An example of a tool being inserted into the middle of a sequence is shown in Figure 5.8.

Figure 5.7. Examples of inserting a new tool at the end of the sequence, when using the Gradual Mutation operator.


Tool A, Tool B, Tool C, Tool D
Tool A
Tool B
Tool C
Tool D
*
Tool A
Tool B
Tool C
Tool D
Tool E
Tool A, Tool B, Tool C, Tool D, Tool E
Tool A, Tool B, Tool C, Tool D
*
Tool A
Tool B
Tool C
Tool D
Tool A
Tool B
Tool C
Tool D
Tool A
Tool B
Tool C
Tool D
Tool E
Tool A, Tool B, Tool C, Tool D, Tool E

Case 1

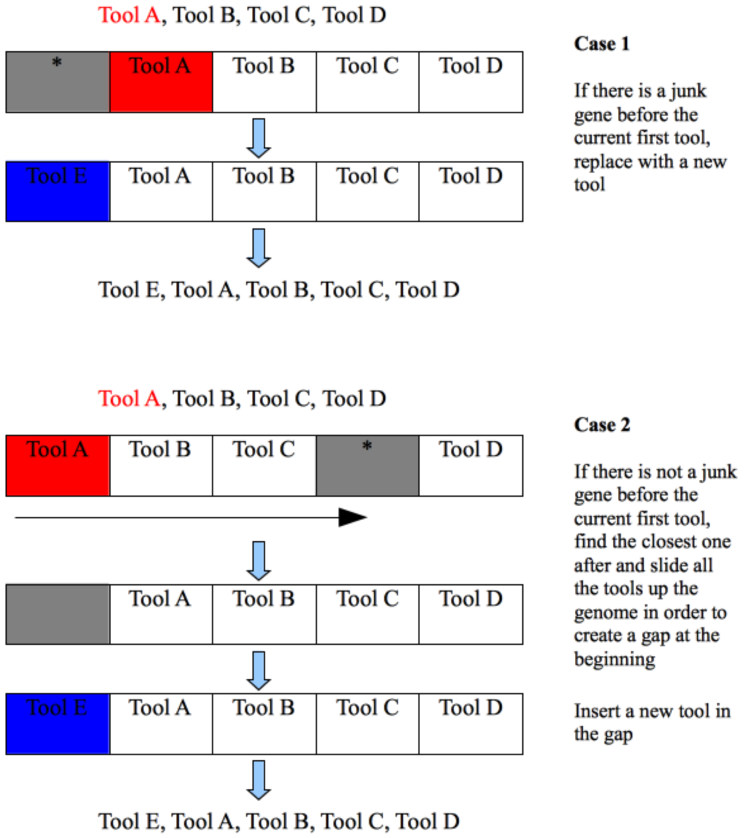
If there is a junk gene after the current last tool, replace with a new tool

Case 2

If there is not a junk gene after the current last tool, find the closest junk gene and slide all the tools down the genome to create a gap at the end

Insert a new tool in the gap

Figure 5.6. Examples of inserting a new tool at the beginning of the sequence, when using the Gradual Mutation operator.


Tool A, Tool B, Tool C, Tool D
*
Tool A
Tool B
Tool C
Tool D
Tool E
Tool A
Tool B
Tool C
Tool D
Tool E, Tool A, Tool B, Tool C, Tool D
Tool A, Tool B, Tool C, Tool D
Tool A
Tool B
Tool C
*
Tool D
Tool A
Tool B
Tool C
Tool D
Tool E
Tool A
Tool B
Tool C
Tool D
Tool E, Tool A, Tool B, Tool C, Tool D

Case 1

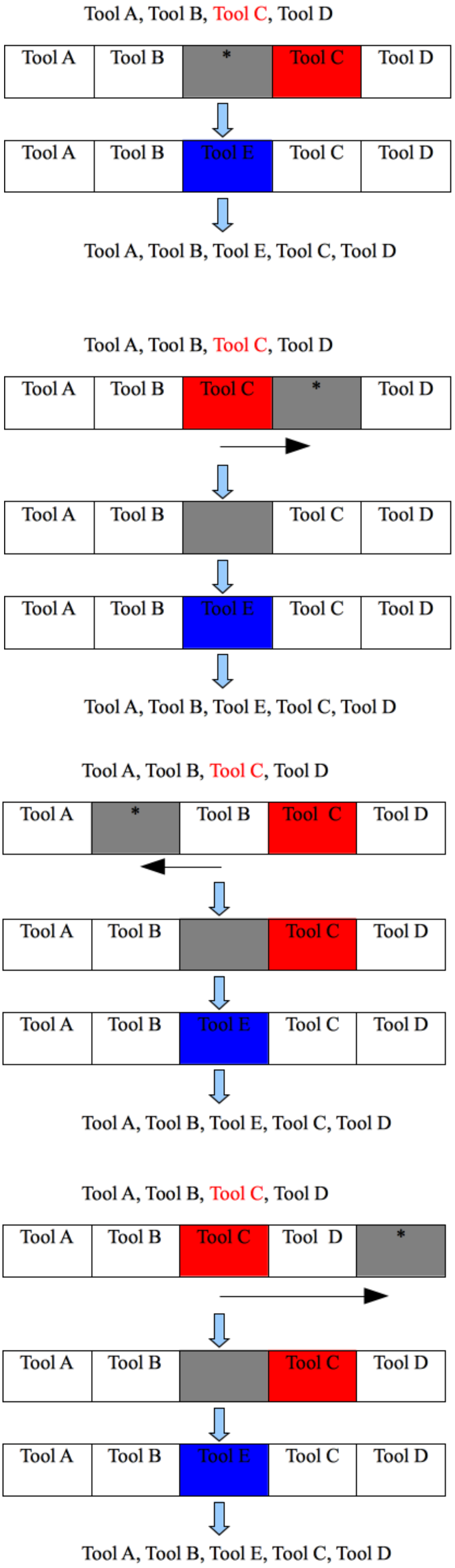
If there is a junk gene before the current first tool, replace with a new tool

Case 2

If there is not a junk gene before the current first tool, find the closest one after and slide all the tools up the genome in order to create a gap at the beginning

Insert a new tool in the gap

Figure 5.8. Examples of inserting a tool into the middle of a sequence, when using the Gradual Mutation operator.



Case 1

If there is a junk gene directly before the chosen gene, replace with a new tool

Case 2

Otherwise, if there is a junk gene directly after the chosen gene, slide the chosen gene up creating a gap.

Insert the new tool into the gap

Case 3

Otherwise, if there is a junk gene before the chosen gene, slide the content of all the genes left of the chosen gene left creating a gap.

Insert the new tool into the gap

Case 4

Otherwise, if there is a junk gene after the chosen gene, slide the content of all the genes after and including the chosen gene left creating a gap.

Insert the new tool into the gap

5.3.1.4 Evaluation of Tool Sequence Solutions

As discussed in Chapter 3, tool sequences are evaluated using the CAM software, Machining Strategist (Vero Software, 2012). This provides a set of tool paths for each tool and a 3d model showing the remaining excess stock. The goal of the optimisation work in this chapter is to find the tool sequence, $\boldsymbol{E}$, that has the shortest machining time and a maximum thickness of excess stock less than the required surface tolerance, c.

The objective function used for a tool sequence, $\boldsymbol{E}$, and part, p, is given by,

$$f_{eval}(\boldsymbol{E}, p) = f_m(\boldsymbol{E}, p) + k$$

$$k = \begin{cases} 2k, \ d > (\mathrm{c} + \varepsilon) \\ k, \quad \mathrm{c} \geq d \leq (\mathrm{c} + \varepsilon) \\ 0, \ d < \mathrm{c} \end{cases} \qquad (5.1)$$

where f_m is a function that calculates the total machining time using tool path lengths (given in equation 3.2 from Chapter 3) and k is a punishment factor determined by the maximum thickness of excess stock, d. k is used to handle the surface tolerance constraint. It is a constant value, and adds machining time to a sequence to make it less attractive if it has not met our constraint, c. A smaller punishment is added if d is within ε of c.

5.3.2 Algorithms

In this section we discuss the implementations of the four metaheuristic algorithms used in the experiments below, as well as presenting the results of parameter tuning experiments.

5.3.2.1 Stochastic Hill Climbing

Hill Climbing (HC) was described previously in Chapter 2. Stochastic Hill Climbing (SHC) is a variant of HC where any improvement is accepted, allowing for a less exhaustive search of the neighbourhood. Here, the algorithm is implemented in the following way, to find a tool sequence to produce a part, p:

```
1. Select a starting tool sequence, E, randomly and evaluate $f_{eval}(\boldsymbol{E}, p)$,
   set evaluations = 1, m = 0
   while (m < 1000)
       2. Mutate E, giving E', until a legal E' is created. Set m = m + 1 for each mutation attempt
       3. Evaluate $f_{eval}(\boldsymbol{E}', p)$ and set evaluations = evaluations + 1
       4. If $f_{eval}(\boldsymbol{E}', p)$ < $f_{eval}(\boldsymbol{E}, p)$, set E to E'. Set the number of unsuccessful mutations, m, to 0.
5. End and return the lowest scoring tool plan found
```

5.3.2.2 Random Restart Stochastic Hill Climbing

5.3.2.2.1 Implementation

Random Restart Stochastic Hill Climbing is another variant of HC. Here, SHC is repeated from a random starting point after a stopping condition is met (Hoos, 1999). Another stopping condition is used to end the search. This modification can assist in escaping local optima. In order to severely restrict the number of evaluations used by the algorithm, these stopping conditions were carefully planned and tested. To keep an individual SHC run short, two conditions were applied:

1. *Stop if more than n unique solutions are evaluated, where n is predetermined*
2. *If more than γ iterations – replacing the current best solution with a higher scoring alternative – have taken place, stop if more than y evaluations occur during an iteration, where γ and y are predetermined*

To restrict the number of restarts required, this condition was applied:

> *Stop if the final member of an individual search run has a fitness value higher than or equal to ρ individuals of the best members archive, **B**, where ρ is predetermined.*

The implementation of the RRSHC algorithm used here works as follows:

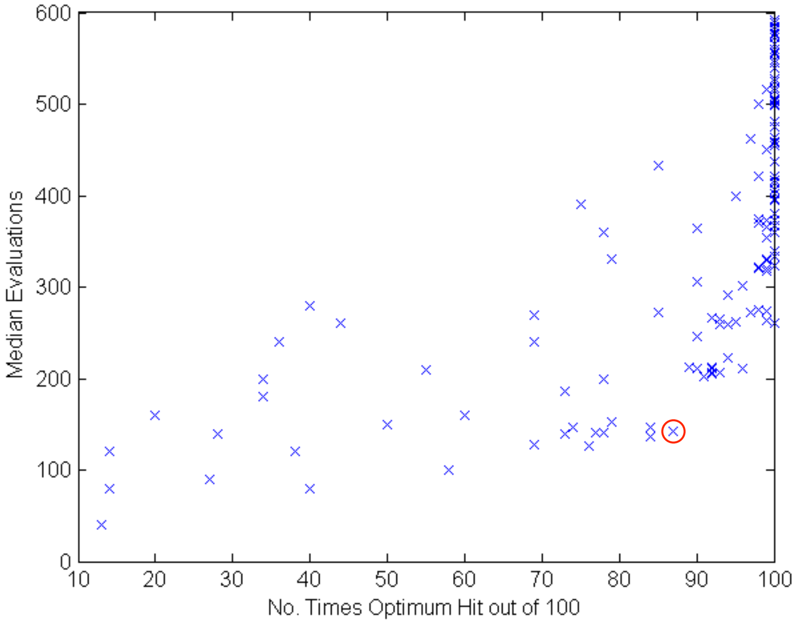
```
1. Set evaluations = 0, m = 0, $n_{count}$ = 0, $\gamma_{count}$ = 0, $y_{count} = 0$, and $\boldsymbol{B} = \emptyset$
   while ($n_{count}$ < n) and ( $\gamma_{count}$ < $\gamma$)
      2. Select a starting tool sequence, E, randomly and evaluate $f_{eval}(\boldsymbol{E}, p)$. Set evaluations = evaluations + 1, $\gamma_{count}$= $\gamma_{count} + 1$
      while (m < 1000) and ( $y_{count}$<y)
         3. Mutate E, giving E', until a legal E' is created. Set m = m + 1 for each mutation attempt
         4. Evaluate $f_{eval}(\boldsymbol{E}', p)$ and set evaluations = evaluations + 1 and $n_{count}$ = $n_{count}$ + 1, if E' has not been evaluated before
         5. If $f_{eval}(\boldsymbol{E}', p)$ < $f_{eval}(\boldsymbol{E}, p)$, set E to E', otherwise set $y_{count}$ = $y_{count}$ + 1.
      6. Add E to B. If $f_{countBetter}(\boldsymbol{E}, p, B)$ > $\rho$, break
7. Sort B and return the highest ranking solution
```

5.3.2.2.2 Parameter Testing

There are three parameters set at the start of the search, γ, n and ρ. These greatly affect the number of evaluations. To find the best values, the algorithm was run for 100 trials on Part 1, with a number of different parameter values. There was a clear trade-off between the number of times the optimum result was found and the number of evaluations used.

Figure 5.9 shows a scatter plot of these two objectives, for every parameter pair tested. There was a clear pattern that results with a higher number of evaluations reaching the optimum more often. The results were then sorted primarily by the number of times that the optimum solution was found and secondarily by the average number of evaluations used. As we have two requirements for our algorithms – finding the best results and using the fewest number of evaluations, the configuration choice was difficult as there was no definitive solution that excelled in both criteria. The results were filtered by requiring that mean evaluations were under 140 (determined by comparing to the number of

Figure 5.9. Plot showing for the different parameter values, the number of times that the optimum was reached out of 100 trials and the median evaluations used.



evaluations required by other algorithms) and that the median fitness was equal to the optimum solution, meaning that at least half of the trials for a particular parameter configuration reached the optimum. The solution circled in Figure 5.9 was chosen for use in the experiments below as it represented a good trade-off between the two. This used the parameter configuration, ρ = 2 and n = 120.

5.3.2.3 Genetic Algorithm

5.3.2.3.1 Implementation

Genetic Algorithms (GA) were introduced previously in Chapter 2. The version implemented for the experiments below uses a (λ, μ)-strategy, where at each generation a population of children are created and replace all of the members of the previous population, apart from the best solution, i.e. an elitist strategy. Rank based selection is used to select parents (Goldberg, 1989). At each iteration, new individuals are created through recombination between two parents and the mutation operator described above.

To optimise the sequence of tools to machine a part p, the Genetic Algorithm used in the experiments below followed this scheme:

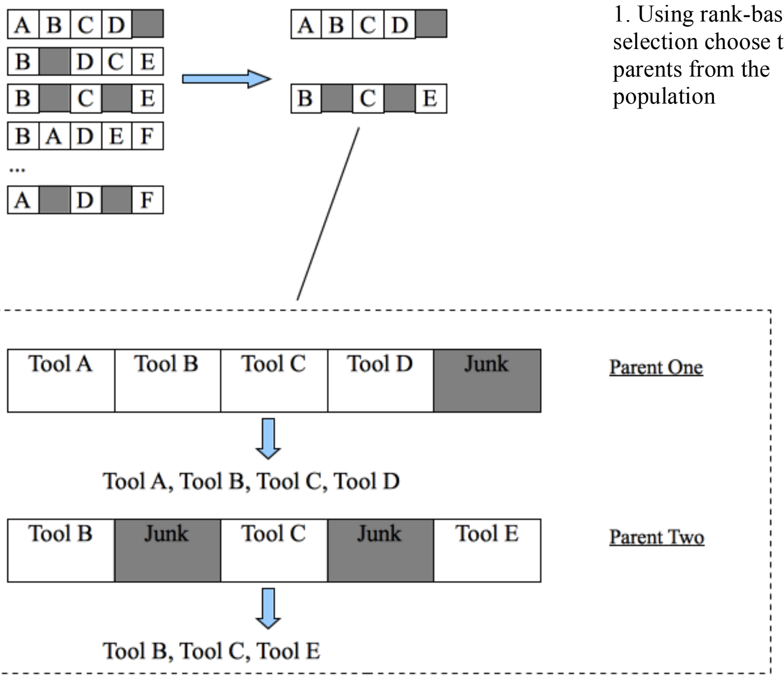
1. *Generate **N**, an initial population of j random individuals, randomly chosen from the entire search space. The population size, j, is predetermined*
2. *For each* $\boldsymbol{E} \in \boldsymbol{N}$, *evaluate* $f_{eval}(\boldsymbol{E}, p)$
3. *Create a new population N', initialised with the singled highest ranked member of N according to* $f_{eval}(\boldsymbol{E}, p)$
4. *Use the crossover method, described below, with a probability of crossover, c, to create a new child,* $\boldsymbol{E}_{child}$.
5. *A mutation is applied to* $\boldsymbol{E}_{child}$ *with probability, m*
6. *Add* $\boldsymbol{E}_{child}$ *to N'*

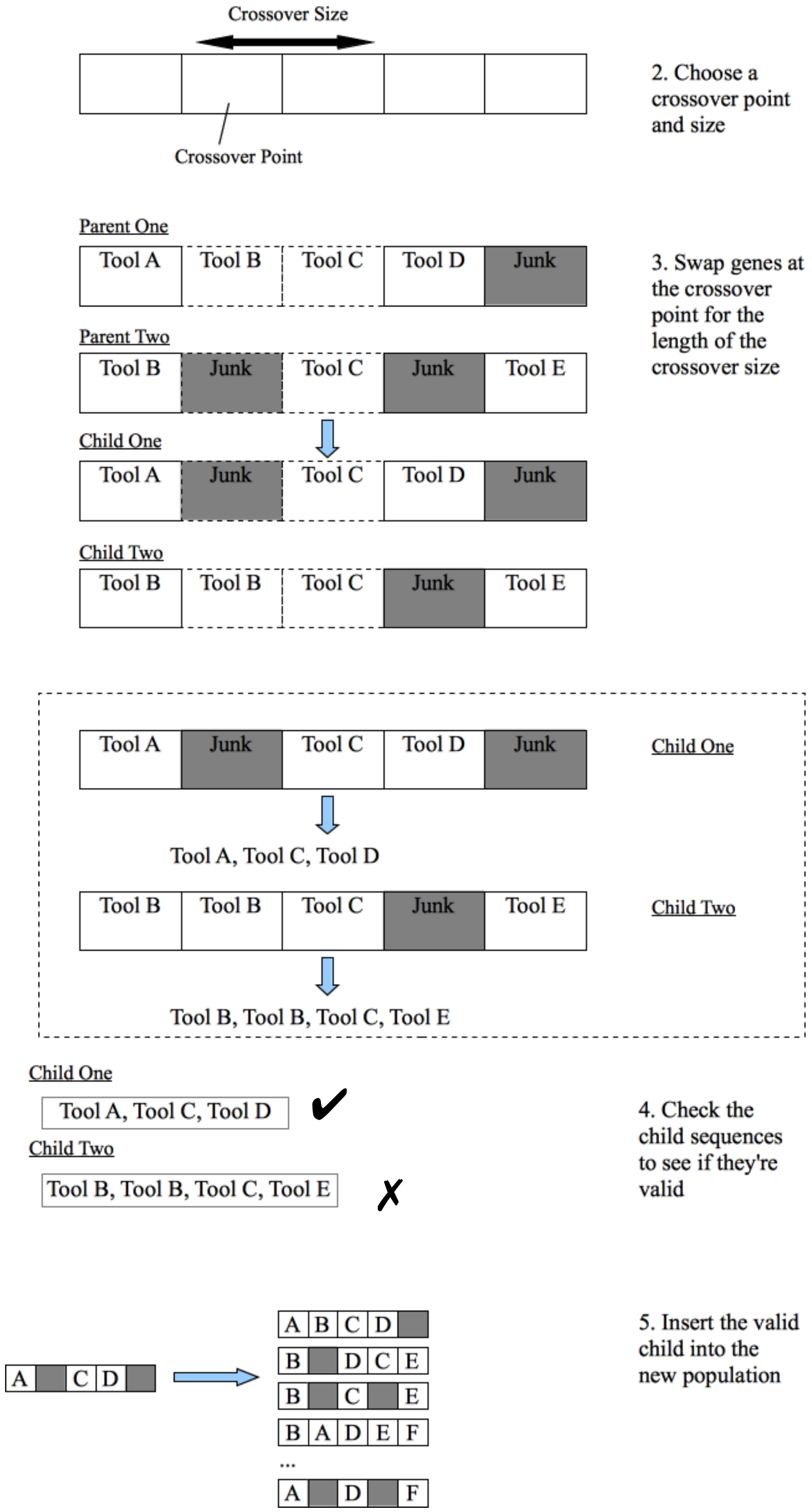
```
7.  If size(N') < j, repeat 4 to 6, otherwise set N = N'
8.  Repeat 2 to 7 until the highest ranked member of N according to $f_{eval}(\boldsymbol{E}, p)$ has been the same
    for e consecutive generations
```

The recombination operator handles the generation of a new individual. An example of the operator is shown in Figure 5.10. First, an individual from the current generation is chosen using Rank-Based Selection (Goldberg, 1989). If a generated pseudo-random number is less than the crossover rate, *c*, a second, different individual is similarly selected. Two children are then created by swapping genes between the parents. The number of genes to swap, the *crossover size*, and the position to swap them, the *crossover point*, are determined randomly. These newly generated children are then tested, using the rules described in Section 4.2 of Chapter 4, to see if they are valid tool sequences. If only one child is valid then it is automatically included in the new population. If both children are valid then one is chosen at random. If neither is valid then the initial individual is selected. The procedure is described below:

```
1.  Select an individual, $\boldsymbol{E}_1$ from the current population, N, using rank-based selection, set
    $\boldsymbol{E}_{new} = \boldsymbol{E}_1$
2.  Set rand = pseudo-random number
    if (rand < crossover rate)
        3. select a second individual, $\boldsymbol{E}_2$ from the current population, N, using rank-based
           selection.
        4. Set the crossover size, CS, to a uniform random integer between 1 and 3. Set the
           crossover point, CP, to a random number between 1 and (L – CP), where L is the
           chromosome size
        5. Replace genes indexed CP to (CP + CS) in $\boldsymbol{E}_1$ with the corresponding genes in $\boldsymbol{E}_2$
        6. Replace genes indexed CP to (CP + CS) in $\boldsymbol{E}_2$ with the corresponding genes in $\boldsymbol{E}_1$
        7. If $\boldsymbol{E}_1$ is a valid solution and $\boldsymbol{E}_2$ is not, set $\boldsymbol{E}_{new} = \boldsymbol{E}_1$. If $\boldsymbol{E}_2$ is a valid solution and $\boldsymbol{E}_1$ is
           not, set $\boldsymbol{E}_{new} = \boldsymbol{E}_2$. If both $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ are valid, set $\boldsymbol{E}_{new}$ equal to a random choice
           between the two
8.  Return $\boldsymbol{E}_{new}$
```

Figure 5.10. The recombination operator used by the Genetic Algorithm




Crossover Size
Crossover Point
2. Choose a crossover point and size
Parent One
Tool A
Tool B
Tool C
Tool D
Junk
Parent Two
Tool B
Junk
Tool C
Junk
Tool E
Child One
Tool A
Junk
Tool C
Tool D
Junk
Child Two
Tool B
Tool B
Tool C
Junk
Tool E
3. Swap genes at the crossover point for the length of the crossover size
Tool A
Junk
Tool C
Tool D
Junk
Child One
Tool A, Tool C, Tool D
Tool B
Tool B
Tool C
Junk
Tool E
Child Two
Tool B, Tool B, Tool C, Tool E
Child One
Tool A, Tool C, Tool D ✔
Child Two
Tool B, Tool B, Tool C, Tool E ✗
4. Check the child sequences to see if they're valid
A C D
A B C D
B D C E
B C E
B A D E F
...
A D F
5. Insert the valid child into the new population

5.3.2.3.2 Parameter Testing

The GA has three main parameters: mutation rate, crossover rate and population size. In order to find the best values to use for these parameters, a range of parameter configurations were tested over 100 trials on Part 1. A parameter sweep is important because it allows for a fair comparison between algorithms by making sure each is tuned to produce the best performance and also helps determine how sensitive the algorithm is to changes in values.

The tested parameter ranges were carefully chosen. Based on a need to keep the number of evaluations low, and preliminary tests, it was decided to use a small population size. GAs with small population sizes are often referred to as micro-GAs (Coello Coello and Toscano, 2001). They are commonly used when it is desirable to keep evaluations low because they have faster convergence times (Ahn and Ramakrishna, 2003). In related research, they have been used for this reason by (Krimpenis and Vosniakos, 2009) on a tool selection problem and on machining parameter optimisation by (Sardiñas et al., 2006). In the experiments below, population sizes were tested from 2 to 10. Preliminary tests with SHC suggested that good solutions could be obtained in under 100 evaluations, so it would not make sense to use more typical GA population sizes of 100 to 200 individuals. Mutation and crossover rates were tested from 0 to 1 in intervals of 0.1. For each experiment, the stopping condition was set to stop after the GA had had the same best solution for 25 generations.

To select the parameters to use in the experiment, the set of configuration results (each tested for 100 trials) were filtered so that candidate configurations had to have achieved a median fitness equal to the optimum solution (i.e. they needed to find the optimum solution on at least half of the trials) and at least 95 of the 100 trials had to return a feasible solution with a machining time under 160 minutes. This corresponds to 35th best solution (out of 39,607), which was considered to be a good result. This ensured that the configuration could achieve good quality solutions. Once this filtering was performed, the remaining configurations were sorted by the average number of evaluations used, enabling the selection of parameters that provided the fastest convergence. The top five configurations according to this selection method had population sizes of seven, nine and ten. The results from all of the parameter values for these populations are seen in Figure 5.11 and Figure 5.12.

The graphs show that there are contrasts in the performances of the different population sizes. Populations of 9 and 10 have much greater average evaluations but these larger populations reach the optimum far more frequently than the population of 7. The final configuration selected used a population of 9, with a mutation rate of 0.4 and a crossover rate of 0.7 as it offered a good trade-off between evaluation cost and solution quality.

Figure 5.11. Results from the parameter sweep for the Genetic Algorithm with a population size of 7 and a population of 9.

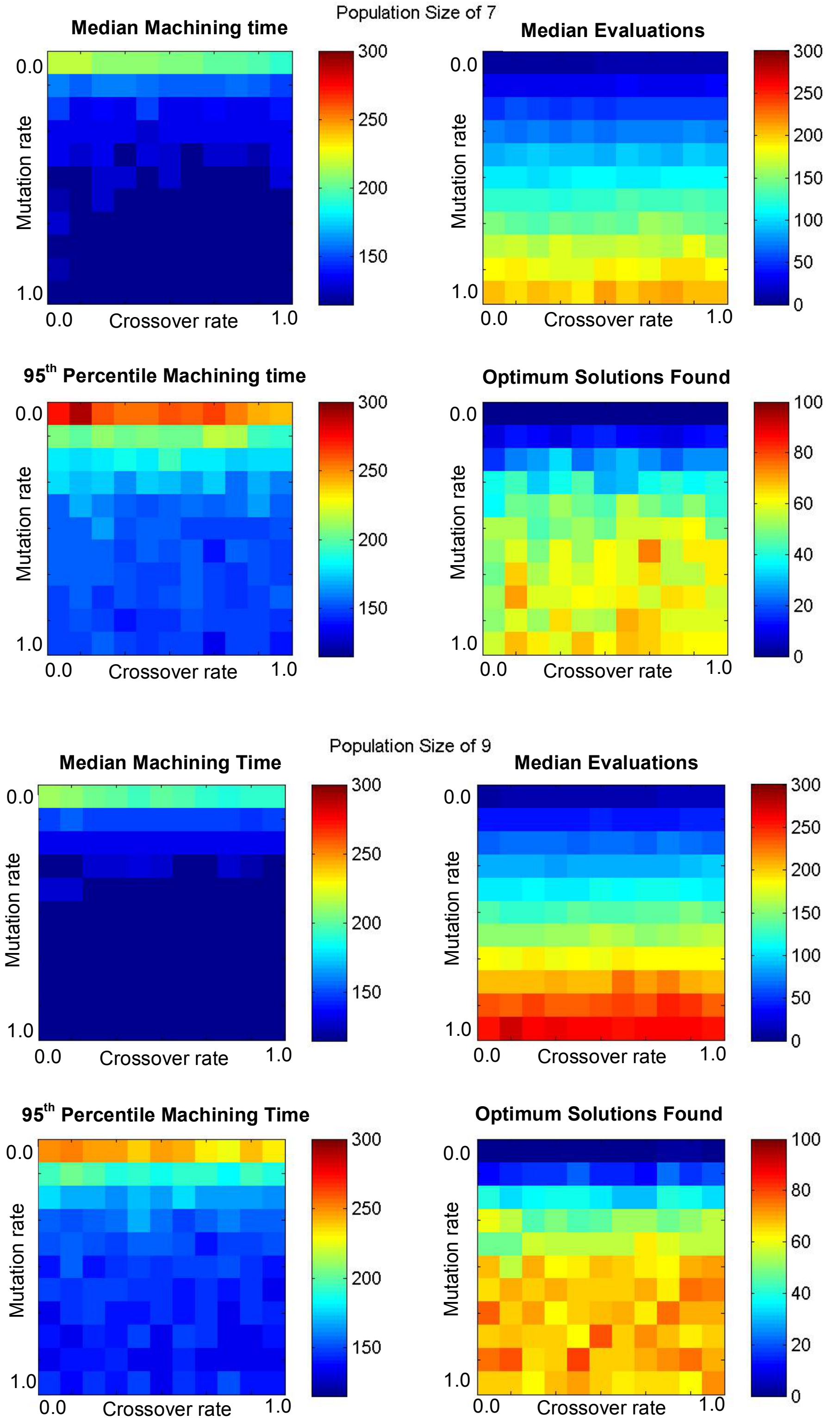
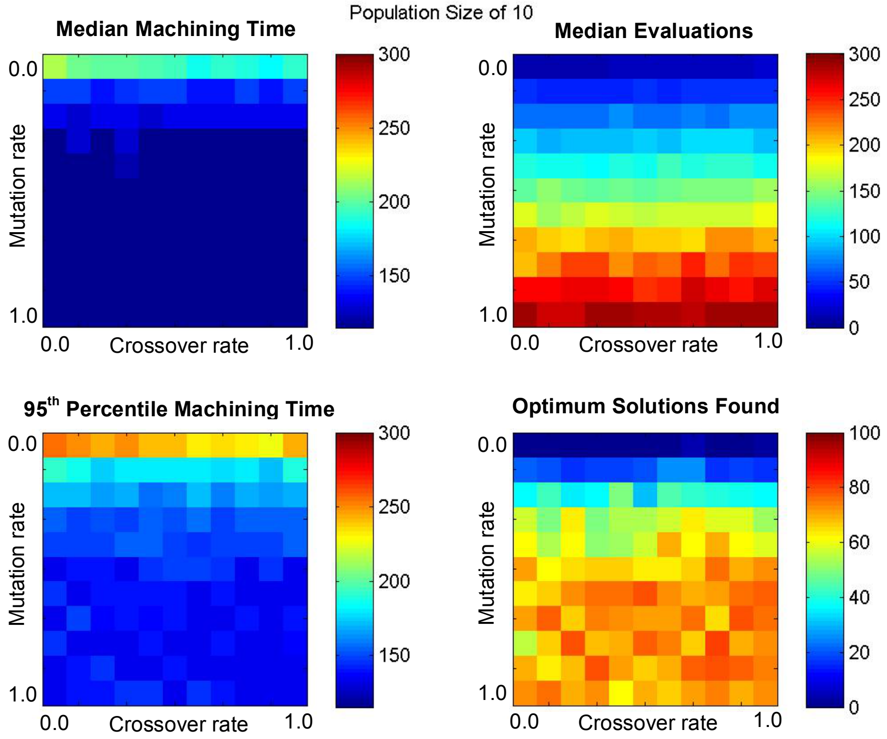


Figure 5.12. Results from the parameter sweep for the Genetic Algorithm with a population of 10.



5.3.2.4 Hybrid Genetic Algorithm

5.3.2.4.1 Implementation

One method that can improve the performance of a GA is including local search during the evolutionary cycle. In this way the GA is being hybridised with another search technique. For example much faster convergence has been shown on the Travelling Salesman Problem by combining a GA with local heuristic search (Sengoku and Yoshihara, 1998). In Machining Parameter Optimisation, (Wang et al., 2005b) hybridised a Genetic Algorithm with Simulated Annealing, while (Baskar et al., 2006) hybridised a Genetic Algorithm with Hill Climbing. Both found that the hybrid algorithms were more successful than the original.

This process of hybridisation in Evolutionary Computation is often referred to as *memetic computing*. There are two main ways that it is implemented, through Lamarckian and Baldwinian inheritance. In Lamarckian inheritance, changes in an individual created through local search can be passed on to offspring, while in Baldwinian inheritance, changes cannot be made directly to the genotype (Le et al., 2009).

A hybrid algorithm (HGA) can be created by combining the GA and SHC implementations described above. In the experiments below, SHC (as implemented above) is applied to the best member of the GA population at various stages in evolution and uses Lamarckian inheritance. The exact implementation, for a part, *p*, works in this way:

1. *Generate **N**, an initial population of j random individuals, randomly chosen from the entire search space. The population size, j, is predetermined* Evaluate and rank each member of the population
2. *Perform SHC search (as described above in Section 5.3.2.1) on the highest ranked member of the population, $\boldsymbol{E}_{best}$, according to $f_{eval}(\boldsymbol{E}, p)$, if starting conditions i and n have been met (see below). If an improvement is found, set $\boldsymbol{E}_{best} = \boldsymbol{E'}_{best}$. If SHC is used, set n_{count} = n_{count} + 1, g_{count} = g_{count} + (number of evaluations used by SHC run)*
3. *Create a new population N', initialised with $\boldsymbol{E}_{best}$*
4. *Use the crossover method, described above, with a probability of crossover, c, to create a new child, $\boldsymbol{E}_{child}$.*
5. *Have a probability, m, chance of mutation applied to $\boldsymbol{E}_{child}$*
6. *Add $\boldsymbol{E}_{child}$ to **N'***
7. *If **N'** < j, repeat 4 to 6, otherwise set **N** = **N'***
8. *For each $\boldsymbol{E} \in \boldsymbol{N}$, evaluate $f_{eval}(\boldsymbol{E}, p)$*
9. *Repeat 2 to 8 until the highest ranked member of N according to $f_{eval}(\boldsymbol{E}, p)$ has been the same for e consecutive generations*

There are many ways that local search can be integrated into a GA. For example, it could be applied to every newly generated individual. However, this extra search is extremely expensive in terms of evaluations, so we use a number of parameters to restrict how often it takes place, and how deep the local search can go. The following parameters are used to restrict the search:

n, the maximum number of times local search can be used

i, the number of consecutive generations that the best member remains the same for before allowing local search to begin

g, the maximum number of evaluations that an individual local search run can use

f, whether or not to restrict local search to flat mutations only

This last parameter, *f*, requires some further explanation. The mutation procedure in both operators described above provides for three changes to a tool sequence:

1. *One tool can be changed in the sequence, keeping the sequence length the same.*
2. *One tool can be removed from the sequence, reducing the sequence length by one tool.*
3. *One tool can be added to the sequence, increasing the sequence length by one.*

What has been termed a *'flat mutation'* is option number 1 only, where no changes can be made to the sequence length. This stops the local search from making drastic changes, which could prove to be disruptive to the global evolutionary search.

5.3.2.4.2 Parameter Testing

As with the other search methods, good parameter values were unknown at the beginning, so configurations were tested for 100 runs each to find a good set to use. The GA parameters were fixed to the parameters found in the GA sweep presented above, using a population size of 9, a mutation rate of 0.4 and a crossover rate of 0.7. The hybrid parameter values tested are shown in Table 5.1.

The results from the parameter sweep showed that average evaluations were relatively low, compared to the GA and RRSHC, with this set up of the search procedure. Due to this, the results from the trials were sorted by mean best score only. The five of the top ranked results from the trials are shown in Table 5.2. These give a good indication of the relative importance of the hybridisation parameters and features that allow for a successful search strategy. The first clear pattern is that all five results use flat mutation for local search, suggesting that this provides more effective and efficient search. Another pattern is seen with parameter *n* - the top 5 configurations use Hill Climbing either 3 or 4 times, with 4 being the highest value available in the parameter sweep. Similarly, the histogram in Figure 5.13(a) shows that the majority of results in the top 100 ranked configurations also use a value of 3 or 4 for *n*. The value used for maximum number of evaluations allowed in the hill climbing runs, *g*, is much more varied. The histogram in Figure 5.13(b) shows no clearly dominating values in the top 100 configurations. The top ranked configuration from Table 5.2 was chosen for use in the experiments below.

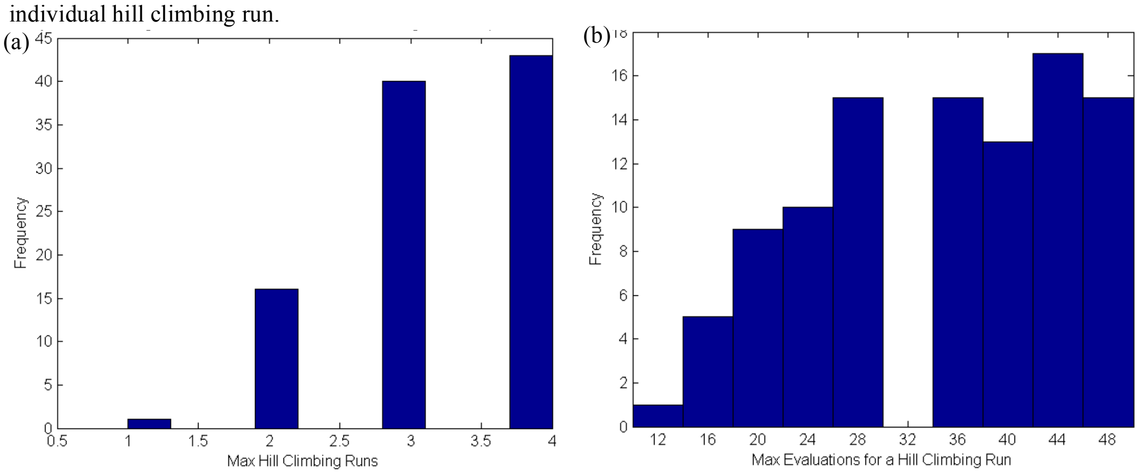
Table 5.1. Range of different parameter values for the Hybrid Genetic Algorithm tested over 100 trials.

Parameter	Lowest value	Highest Value	Interval
n	1	4	1
i	0	8	2
g	10	50	5
f	0 (False)	1 (True)	1

Table 5.2. Showing the top 5 ranked parameter combinations (over 100 trials) from the parameter sweep for the hybrid algorithm.

Rank	n	i	e	f	Mean best	Mean Evaluations	# Optimum Solutions
					Score		Found
1	3	6	30	True	117.3	100	93
2	4	0	30	True	117.7	87.5	93
3	4	2	35	True	117.9	91	90
4	3	6	40	True	117.9	98.5	90
5	4	2	50	True	118.1	89	89

Figure 5.13. Histograms showing the distribution of (a) the maximum individual hill climbing runs parameter value used in the top 100 hybrid algorithm configurations and (b) the maximum evaluations allowed in an individual hill climbing run.



5.4 Results

In this section, we first test the four metaheuristic algorithms using the Global Mutation operator on Part 1. We then test the same algorithms on the same part but using the Gradual Mutation operator and compare the differences in performance. Part 1 was chosen because it is small in size, with a short optimal sequence but has a multimodal search space. In a third experiment, the algorithms are further tested on three additional parts. These are Part 2, which is much larger in size but again has a short optimal sequence; Part 3, chosen because it has a longer optimal sequence; and Part 2-4-H, which has a deceptive search space created by simulation errors, testing the effectiveness of the algorithms. After presenting the results of these experiments, the search spaces of the four tested parts are analysed, and the stopping conditions are extended to allow the GA and RRSHC further time to explore the difficult search space of Part 2-4-H. All experiments use the 18-tool library presented in Table 4.1 from Chapter 4, and the parts used (apart from 2-4-H) are presented in Section 3.7 of Chapter 3. As the algorithms are stochastic and we are looking for a consistently good performance for use in real world situations, each experiment is repeated 10,000 times. This is achieved by caching the whole search space between 1 and 5 tools for each of the four parts.

The motivation behind the experiments in this chapter is to explore the use of metaheuristic algorithms for automating the process of tool selection in real world rough milling operations. To assist a human operator, search needs to be both quick and successful. With this in mind, the search algorithms are assessed both on their ability to find high performing tool sequences and the average run time of the search process in terms of the number of unique evaluations required.

5.4.1 Global Mutation Operator

5.4.1.1 Experiment

In the first experiment, the four algorithms are applied to optimise a tool sequence of between one and five tools using Part 1 and the 18-tool library described in Chapter 4. The algorithms are run until their stopping conditions are met, using a cached version of the search space. In this experiment the algorithms all use the Global Mutation operator.

5.4.1.2 Results

Table 5.3 shows the results of the metaheuristic algorithms' performance over 10,000 trials and the histograms in Figure 5.14 show the distributions of (a) scores, by which we mean total machining times with punishment factor, and (b) evaluations used over the 10,000 runs. All of the search methods could be considered successful in tackling this task. Each of them achieves a mean machining time close to the optimal solution and they are all able to find good solutions after surveying only a small section of the search landscape. However, there are many differences between the performances of the individual algorithms.

There is no clear winner, an algorithm that finds the best solution with the fewest evaluations. It can clearly be seen that hybridising the GA has a very positive effect and improves upon the GA search at

Table 5.3. Results from the Global Mutation operator experiment on Part 1. A Kruskal-Wallis test for equal distributions showed that there were significant differences between at least two algorithms ($p < 1 \times 10^{-9}$). Post-hoc analysis using a pairwise Wilcoxon rank sum test for equal distributions and the Bonferonni correction found significant differences between all algorithms ($p < 1 \times 10^{-9}$).

Algorithm	Best Score (mins)	Worst Score (mins)	Mean Score (mins)	95th Percentile Score	Median Evaluations	95th Percentile Evaluations	Number of times the optimum solution was found
SHC	115.99	245.54	122.43	134.73	88	133	7417
RRSHC	115.99	154.28	117.42	133.24	210	316	9182
GA	115.99	195.65	125.56	151.88	111	168	5785
HGA	115.99	178.35	119.56	134.73	102	142	8214

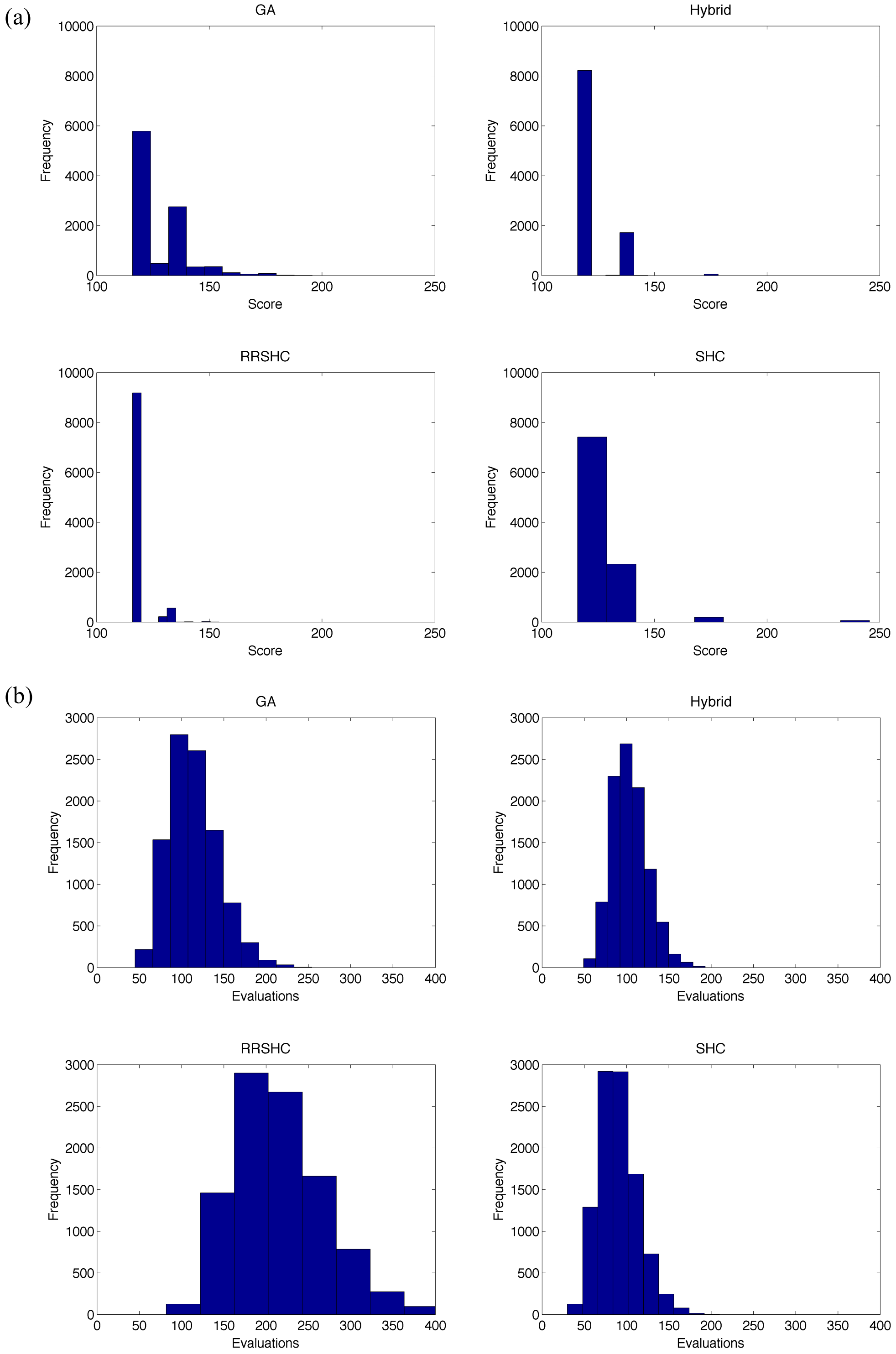
every level. The frequency that the optimum solution is found is raised by 42% while there is a 15% decrease in the number of evaluations in 95th percentile case.

SHC search is the fastest, needing fewer evaluations than the other algorithms. It uses 14% fewer evaluations than the next fastest algorithm, HGA, in the median case. However, the histograms in Figure 5.14(a) show that the algorithm finds worse solutions (e.g. solutions that have machining times of over 150 minutes) more frequently than the others. It is the only algorithm that returns a solution of 246 minutes, which is considerably more than the GA's worst, of 196 minutes. This implies that the SHC algorithm may be more prone to getting stuck in suboptimal areas of solution space.

RRSHC is by far the most successful in terms of finding the optimum solution. This is found around 92% of the time, almost 12% more than HGA, although the 95th percentile scores are similar for both algorithms. However, the chosen configuration of RRSHC uses many more evaluations compared to the other algorithms, with median and 95th percentile evaluations being almost double those used by HGA.

In Figure 5.14(a) we see that all of the algorithms appear to get stuck at similar suboptimal solutions but that RRSHC and HGA clearly outperform the other two. The distributions for evaluations, seen in Figure 5.14(b), are more normal (bell-shaped) than for machining times. RRSHC frequently uses a much larger number of evaluations than the other algorithms, regularly employing more than 250, and in the worst case almost 400. We can also see that SHC has a much larger proportion of runs using under 100 evaluations than the GA or hybrid, which both have a similar distribution.

Figure 5.14. Histograms showing the distribution of (a) the scores, which are the total machining times in minutes with punishment, and (b) the evaluations used by the algorithms over 10,000 runs on the Global Mutation operator experiment on Part 1.


(a)

(b)

5.4.2 Gradual Mutation

5.4.2.1 Experiment

In this second experiment, the four algorithms (GA, HGA, RRSHC and SHC) perform the same task under the same conditions as in 0 but this time use the Gradual Mutation operator. The algorithms are tested over 10,000 individual trials on a cached version of the search space. These results are then compared to those found using the Global Operator in the first experiment.

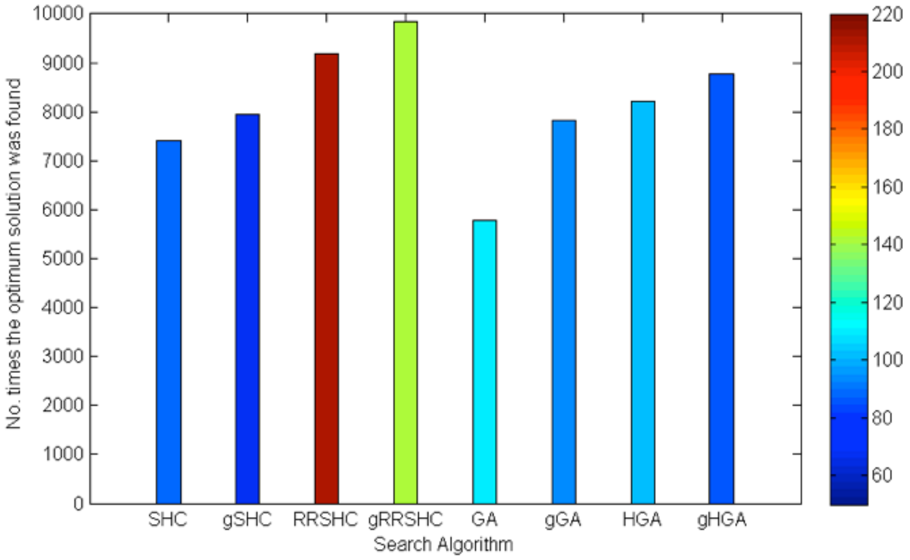
5.4.2.2 Results

Table 5.4 shows the results obtained when using the Gradual Mutation operator, and it can clearly be seen that there is an improvement in performance from all of the algorithms compared to the Global Mutation operator. There is an improvement in both total machining times and unique evaluations across all of the algorithms, which is shown in the bar chart in Figure 5.15. A pairwise Wilcoxon rank sum test for equal distributions was applied, and it found the results are significantly different, with p <

Table 5.4. Showing results from the Gradual Mutation experiment on Part 1. Brackets on Median evaluations and Number of times the optimum solution was found indicate differences from Experiment 1. All of the algorithms were found to be significantly different in terms of scores and evaluations when using a pairwise Wilcoxon rank sum test for equal distributions and the Bonferroni correction for multiple comparisons ($p < 1 \times 10^{-9}$, following a Kruskal-Wallis test indicating there were significant differences between at least two groups), apart from the scores of the GA and SHC, with p = 0.19.

Algorithm	Best Score (mins)	Worst Score (mins)	Mean Score (mins)	95th Percentile Score	Median Evaluations	95th Percentile Evaluations	Number of times the optimum solution was found
SHC	115.99	178.35	119.96	134.73	68 (-24%)	101	7942 (7%)
RRSHC	115.99	134.73	116.23	115.99	141 (-34%)	207	9853 (7%)
GA	115.99	183.63	120.6	134.73	94 (-17%)	139	7834 (35%)
HGA	115.99	178.35	118.4	134.73	87 (-14%)	122	8761 (7%)

Figure 5.15. Bar chart showing the number of times that the optimum solution was found by the four metaheuristic algorithms and their gradual mutation variants (labelled with a 'g' in front of the algorithm name). The colour of the bars shows the number of evaluations used, assigned according to the colour bar on the right. The bar chart compares the results from the first (global mutation) and second (gradual mutation) experiments on Part 1.



1 x 10^{-9} for all four of the algorithms, when comparing the results of the two mutation operators. As can be seen in Figure 5.15, the relative differences between the algorithms remain the same for the most part. For example, RRSHC and the HGA still found the optimum more often and SHC performed the fastest.

The most dramatic differences are the 35% increase in the number of times the GA found the optimum solution and the 32% reduction in evaluations used by RRSHC. Another effect is that the worst solution found is better for all the algorithms apart from HGA, where it remained the same. This is most improved in SHC. Interestingly, Gradual Mutation means that in this experiment, the GA and SHC are not significantly different in terms of machining times, with $p = 0.19$ using a Wilcoxon rank sum test.

5.4.3 Other Parts

In a third set of experiments, the four algorithms are tested to see if similar results are achieved when optimising tool sequences on different parts. Parts 2, 3 and 2-4-H, described in Section 3.7 from Chapter 3, are used for testing. All three parts are larger than Part 1, allowing us to see if changing the dimensions of the component affects the performance of the algorithms. In addition Part 2-4-H has problems in certain discrete areas of the search space, created by an error in the simulation process, allowing us to test the algorithms on a more difficult search space. This is described further in Section 5.4.5 below. The same parameter settings and tool library are used as in the two experiments above, and again, each algorithm is tested across 10,000 trials. In each experiment the algorithms use the Gradual Mutation operator.

5.4.3.1 Part 2

Interestingly, the optimum solution on Part 2 is the same as on Part 1. It is made up of {12.0x0.0(endmill) – 6.0x0.5(toroidal)} but takes considerably longer, 979 minutes, as the part is much larger. The second best solution is different to Part 1, consisting of {20.0x0.0(endmill) – 12.0x0.0(endmill) – 6.0x0.5(toroidal)}, with a total machining time of 1,000 minutes. Table 5.5 shows the results for the four algorithms on this part. All of the algorithms perform well, and improve on their performance on Part 1. RRSHC and SHC are the best performing, with RRSHC finding the optimum solution almost every time. SHC locates the optimum in 98.7% of trials, which is more than HGA, and a 24% improvement on Part 1. This suggests that the search landscape is smooth with many routes to the optimum solution. The histograms in Figure 5.16(a) show that SHC and HGA have a very similar distribution of scores, while the GA terminates at many more suboptimal solutions. However, the hybrid algorithm achieves a lower mean score than SHC, as it often gets very close to the optimal solution, even if it does not reach it.

In terms of evaluations, all of the algorithms apart from RRSHC tend to use fewer than 100 evaluations. RRSHC uses many more, with a reasonable proportion occurring over 200 evaluations. As with Part 1, the evaluations distributions are bell shaped. The GA and HGA share a similar distribution, with SHC skewed to the left side and RRSHC much broader than the others. The evaluations used on average by each algorithm is very similar to Part 1.

Figure 5.16. Histograms showing the distribution of (a) the scores (total machining time in minutes with the punishment factor) and (b) the evaluations obtained over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 2.

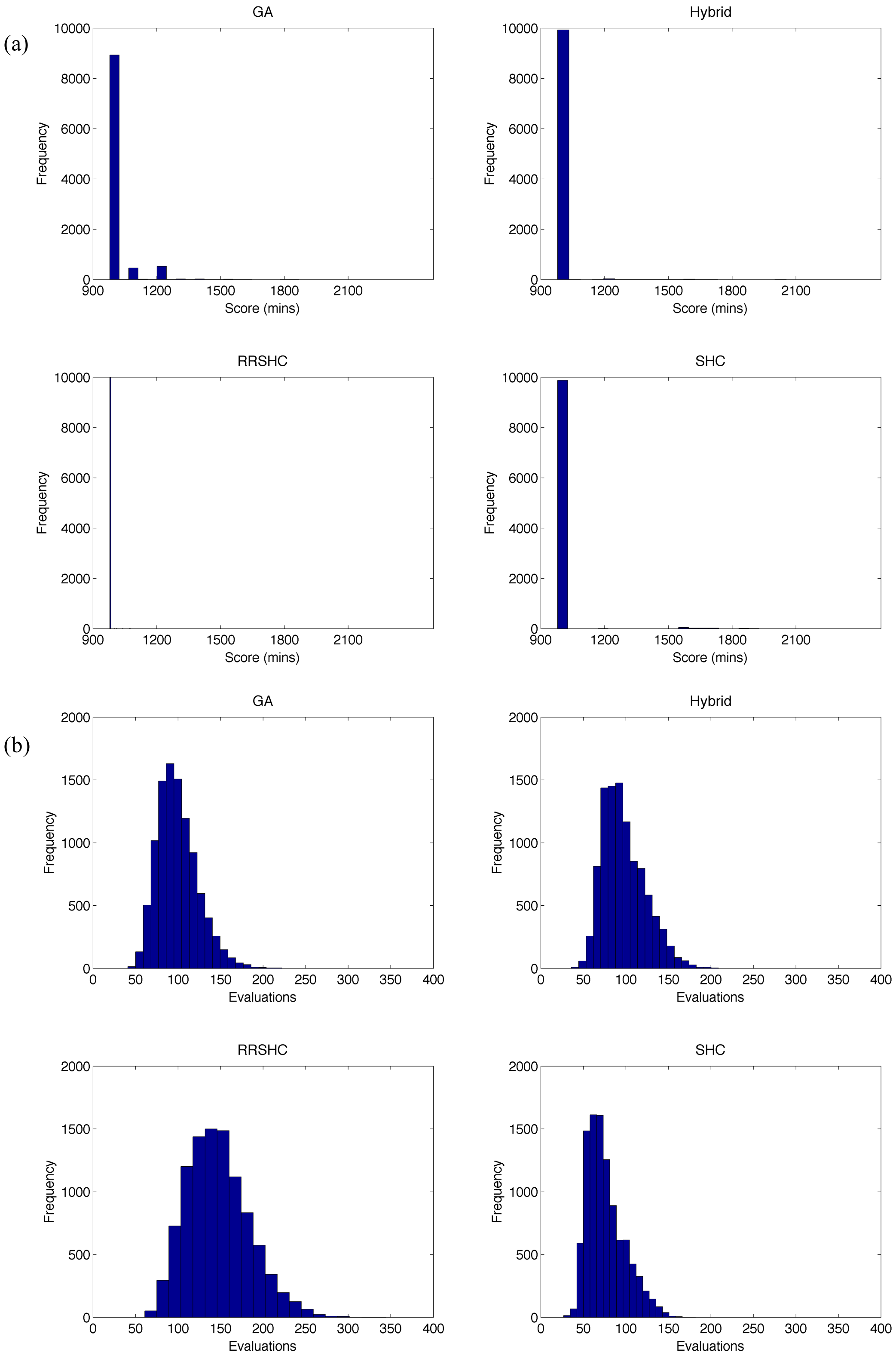


Table 5.5. Results from the third set of experiments, testing four algorithms with the Gradual Mutation operator on Part 2. A Kruskal-Wallis test for equal distributions showed that there were significant differences between the algorithms ($p < 1 \times 10^{-9}$). Post-hoc analysis using a pairwise Wilcoxon rank sum test for equal distributions and the Bonferonni correction found significant differences between all algorithms ($p < 1 \times 10^{-9}$).

Algorithm	Best Score (mins)	Worst Score (mins)	Mean Score (mins)	95th Percentile Score	Median Evaluations	95th Percentile Evaluations	Number of times the optimum solution was found
SHC	979.4	1926.7	988.1	979.4	71	120	9872
RRSHC	979.4	1076	979.4	979.4	143	213	9989
GA	979.4	1868.4	1000.9	1204.3	97	143.5	8472
HGA	979.4	2054.7	982.8	979.4	93	144	9676

Table 5.6. Results from the third set of experiments, testing four algorithms with the Gradual Mutation operator on Part 3. A Kruskal-Wallis test for equal distributions showed that there were significant differences between the algorithms ($p < 1 \times 10^{-9}$). Post-hoc analysis using a pairwise Wilcoxon rank sum test for equal distributions and the Bonferonni correction found significant differences between all algorithms ($p < 1 \times 10^{-9}$), apart from RRSHC and SHC which scored $p = 0.036$.

Algorithm	Best Score (mins)	Worst Score (mins)	Mean Score (mins)	95th Percentile Score	Median Evaluations	95th Percentile Evaluations	Number of times the optimum solution was found
SHC	212.27	1116.27	213.94	212.27	84	125	9979
RRSHC	212.27	238.22	212.31	212.27	168	250	9963
GA	212.27	421.42	216.62	231.69	103	153	7763
HGA	212.27	212.27	212.27	212.27	114	157	10000

Figure 5.17. Histograms showing the distribution of scores (total machining time in minutes with punishment factor) obtained over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 3.

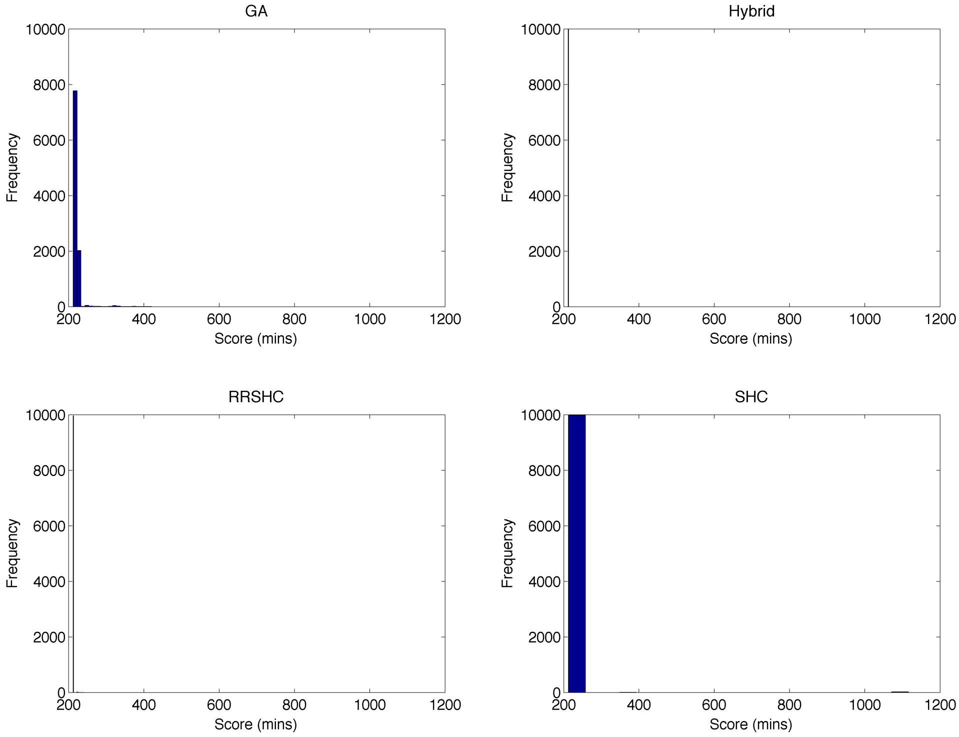
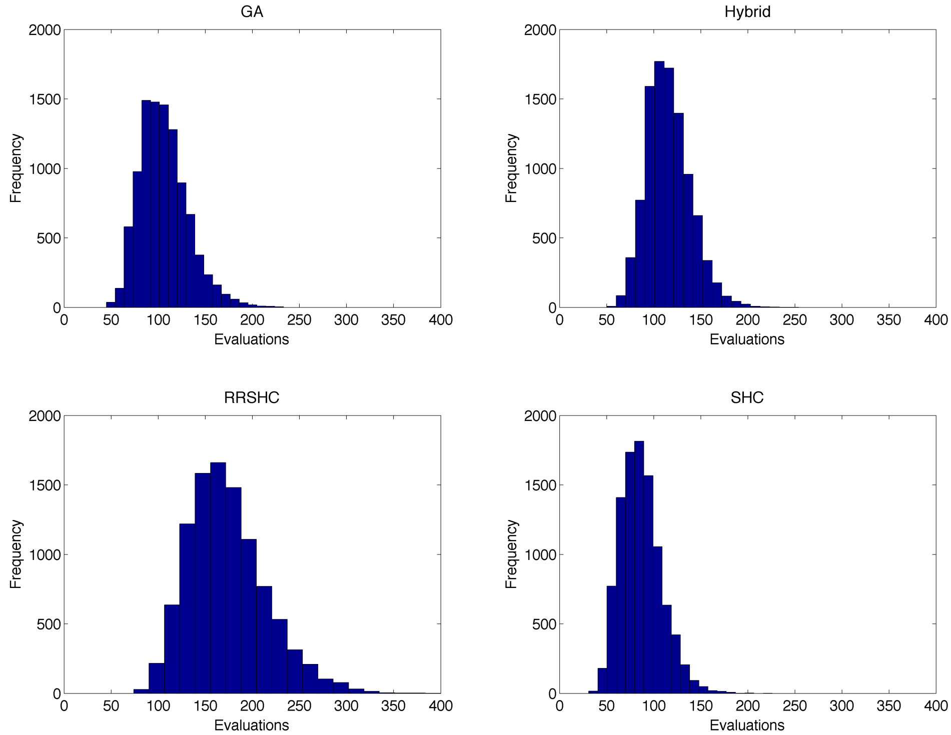


Figure 5.18. Histograms showing the distribution of evaluations achieved over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 3.



5.4.3.2 Part 3

The optimum solution on Part 3 is a three tool sequence consisting of {20.0x0.0(endmill) – 12.0x2.0(toroidal) – 6.0x0.5(toroidal)}, which takes 212 minutes to machine. The HGA, SHC and RRSHC find the optimal solution consistently, with HGA displaying a perfect performance on this part, and a 14% improvement on its performance on Part 1. However, it uses more evaluations on average than SHC or the GA. In terms of total machining time, SHC and RRSHC perform very similarly, and are not significantly different, which represents a large 26% improvement in the performance of SHC compared to the results obtained on Part 1. The histogram in Figure 5.17 does suggest that SHC occasionally finds bad solutions, which RRSHC does not. Similarly to Part 1 and Part 2, the GA gets stuck in suboptimal solutions. However, the majority of those are close to the optimal solution. The distribution of evaluations seen in Figure 5.18 is similar to Part 2. RRSHC has a large number of runs with over 200 evaluations. The majority of SHC runs use fewer than 100 evaluations. Compared to Part 1, the algorithms use a relatively larger number of evaluations, with HGA displaying a 31% and SHC a 23% increase.

5.1.1.1 Part 2-4-H

Part 2-4-H has an optimal solution of {12.0x0.0(endmill) -- 12.0x2.0(toroidal) -- 8.0x4.0(ballnose)}, which takes 1153 minutes to machine. The algorithms find it much more difficult to optimise tool sequences on this part, compared to the other three. RRSHC performs the best, finding the optimum solution on 22% of runs. SHC performs much worse, locating the optimum on only 6% of runs. The histograms in Figure 5.19 shows that the algorithms regularly terminate with many more unique

Table 5.7. Results from the third experiment, testing four algorithms with the Gradual Mutation operator on Part 2-4-H. A Kruskal-Wallis test for equal distributions showed that there were significant differences between the algorithms ($p < 1 \times 10^{-9}$). Post-hoc analysis using a pairwise Wilcoxon rank sum test for equal distributions and the Bonferonni correction found significant differences between all algorithms ($p < 1 \times 10^{-9}$)

Algorithm	Best Score (mins)	Worst Score (mins)	Median Score (mins)	Mean Score (mins)	95th Percentile Score	Median Evaluations	95th Percentile Evaluations	Optimum found
SHC	1152.58	3930.1	1286.42	1311.26	1452.3	71	109	614
RRSHC	1152.58	1496.6	1199.87	1208.72	1308.48	202	323	2238
GA	1152.58	2282.05	1199.87	1252.5	1452.3	101	161	1295
HGA	1152.58	3319.58	1218.80	1263.23	1452.3	95	145	1150

Table 5.8. Showing the top five most frequent scores (*s*) and their frequency of occurrence (*f*) returned over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 2-4-H.

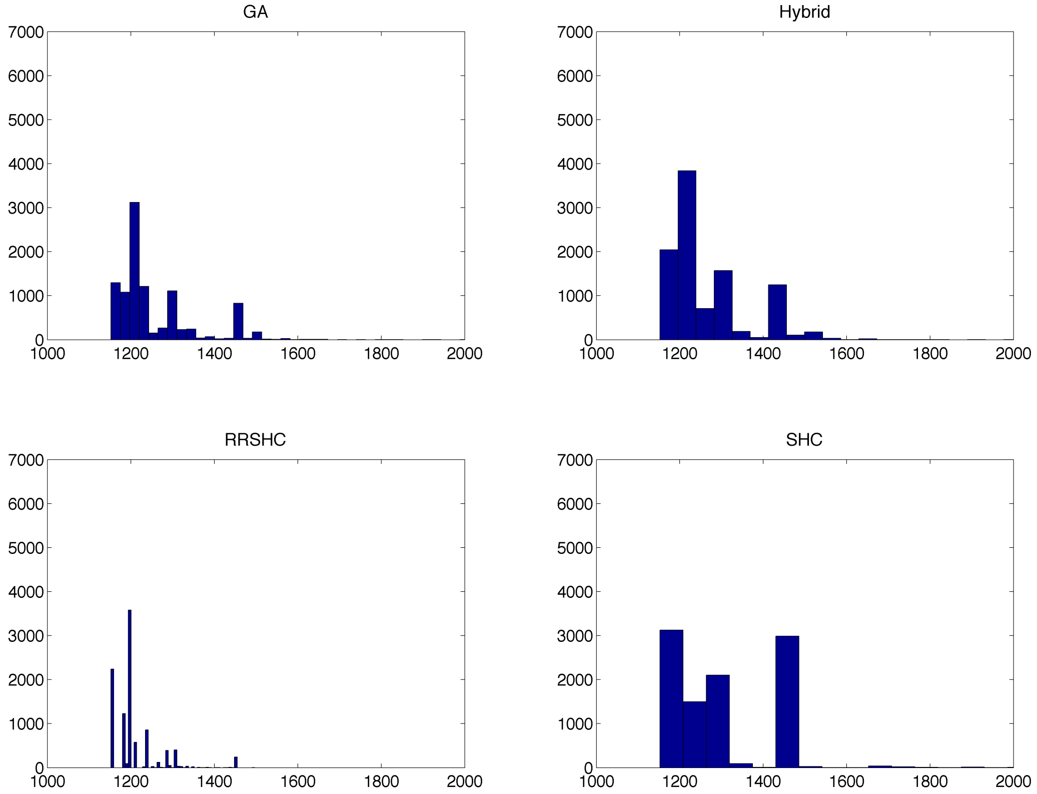
Rank	SHC		RRSHC		GA		HGA	
	s	*f*	*s*	*f*	*s*	*f*	*s*	*f*
1	1452.3	2909	1199.9	3580	1199.9	2660	1199.9	2519
2	1199.9	2047	1152.3	2238	1152.6	1295	1452.3	1212
3	1308.5	1189	1182.8	1214	1182.8	819	1152.6	1150
4	1152.6	614	1209.7	578	1457.3	800	1308.5	999
5	1286.4	480	1237.4	433	1237.4	410	1182.8	836

solutions than with the other parts, indicating that there are many local optima contained in this search space.

In Table 5.8 we see the top five most frequently attained scores for each algorithm. Three of the four algorithms most often find the solution {16.0x0.0(endmill) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose)}, with a machining time of 1199.9 minutes. SHC differs from the other three, most often finding the solution {16.0x0.0(endmill) – 16.0x2.0(toroidal) – 6.0x0.0(endmill) – 16.0x8.0(ballnose) – 8.0x4.0(ballnose)}, with a machining time of 1452.7 minutes. The machining time distributions in Figure 5.19 show that the other algorithms also frequently find solutions in this region of 1400 – 1500 minutes, although this is greater in the HGA, which suggests that it could be drawn into suboptimal space by the local search. The GA and HGA have similar distributions of machining times but the GA more frequently find solutions with machining time under 1200 minutes, leading to a lower median score. RRSHC and the GA share the same median score of 1199.9 minutes. However, the GA finds many more suboptimal solutions between 1300 and 1500 minutes. HGA and SHC find much lower worst scores than the other two algorithms. SHC's worst score is 3930 minutes, which is over double that found by RRSHC in the worst case. This could indicate that SHC can be a bad method to use when there are many local optima.

Figure 5.20 shows the distributions of evaluations used by the algorithms. The majority of SHC runs use fewer than 100 evaluations, considerably less than the other algorithms. This suggests that SHC is the fastest of the algorithms but could also mean that it gets stuck in local optima after surveying only a small number of solutions, which is indicated by the number of runs that terminate after fewer than 50 evaluations. There is a larger distribution of runs with fewer than 50 evaluations than with Part 3 (seen in Figure 5.18). As in the other experiments in this chapter, RRSHC uses many more evaluations, which could be a reason for its relative success on this part. It regularly uses more than 250

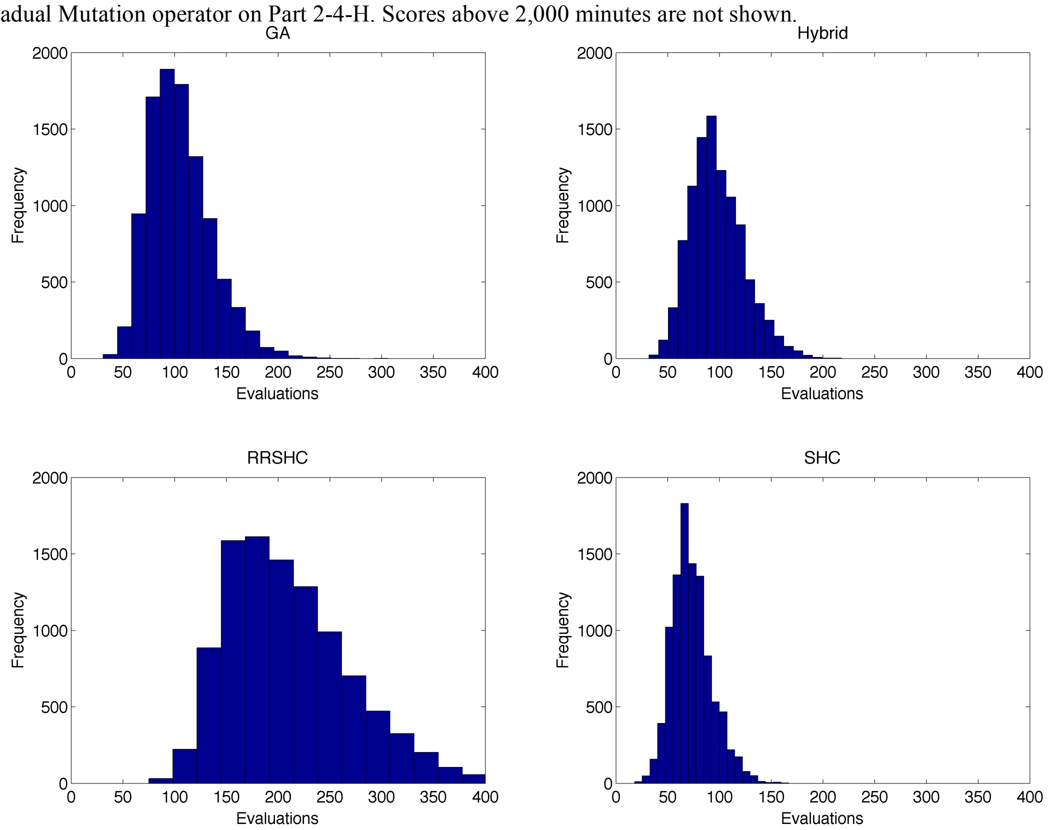
Figure 5.19. Histograms showing the distribution of scores (total machining time in minutes with punishment factor) achieved over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 2-4-H. Scores above 2,000 minutes are not shown.



evaluations, and around 10% of runs employ more than 300 evaluations. The GA and HGA use around 100 evaluations on average, with the hybrid often needing slightly less than the GA. The large number of extra evaluations is likely to be a reason for why RRSHC outperforms the GA and HGA.

While the algorithms were not able to reach the optimum solution as consistently on Part 2-4-H as they could with the others, they were still able to find good solutions. Putting the results in context, out of approximately 40,000 solutions, the mean value for total machining time corresponds to results within the top 8 for RRSHC, 28 for the GA, 35 for the HGA and 85 for SHC. This shows that the algorithms can be used to successfully automate the tool selection problem even when the search space is very complex and difficult to navigate as with Part 2-4-H.

Figure 5.20. Histograms showing the distribution of evaluations used over 10,000 runs by the algorithms using the Gradual Mutation operator on Part 2-4-H. Scores above 2,000 minutes are not shown.



5.4.4 Expanding on Part 2-4-H

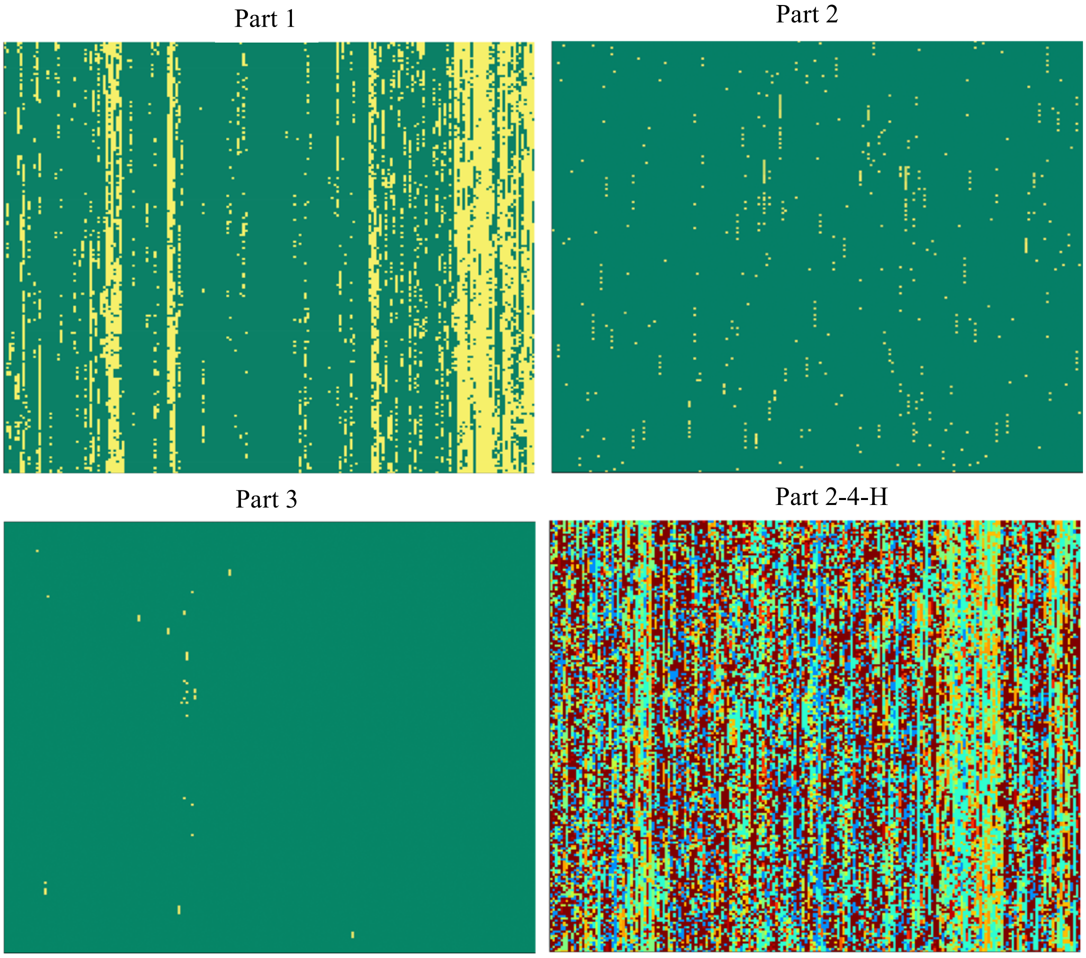
In this section we look at the search spaces of the four different components and discuss reasons for the algorithms' poorer performance on Part 2-4-H, in terms of finding the optimum solution. We then expand the stopping conditions and exploration depth of the GA and RRSHC to see how this improves the ability of the algorithms to consistently locate the optimum on this part and what the extra evaluation costs are.

5.4.4.1 The Search Spaces of the Different Components

The experiments in this chapter have shown that while the metaheuristic algorithms are able to find good solutions on all parts, optimal solutions are discovered much more easily on some parts than others. The algorithms performed particularly well on parts 2 and 3. They performed quite well on Part 1 but found Part 2-4-H very difficult to traverse. On this part it was particularly difficult for them to locate the optimum solution.

In order to get a better feel for the search spaces of the four parts, Steepest Ascent Hill Climbing (SAHC) (Russell and Norvig, 2003) was applied to each of the parts. Here, we start with an initial solution, $\boldsymbol{x}$. During each iteration of the algorithm, every neighbour of $\boldsymbol{x}$ is evaluated and the one with the largest improvement is selected. For each of the four parts, this algorithm was run using every valid solution as a starting position, and the same cached solutions used in the results presented above. The results from this can be seen in the colour plots in Figure 5.21.

Figure 5.21. Pseudo-colour plots showing the fitness of the best solution found using Steepest Ascent Stochastic Hill Climbing from every starting position, for parts 1, 2, 3 and 2-4-H. In the plot for Part 2-4-H, dark blue shows the optimum solution was found and red shows a bad scoring solution. In the others, dark green shows the optimum solution was found.



For parts 1, 2 and 3 only two solutions are found by the algorithm, the optimum solution shown in dark green and a suboptimal solution in yellow. This implies that these search spaces consist of two “hills”. The prevalence of yellow coincides with how difficult the majority of the algorithms found the search space. Part 3 has the least yellow and was the only component where one of the algorithms found the optimum solution on every trial. Part 1 was more difficult to traverse than Part 2, and there is a much higher proportion of yellow on the colour plot.

The colour plot for Part 2-4-H is completely different to the others. Here, the optimal solution is in dark blue, while time consuming solutions are shown in red. Analysing the results, we see that the algorithm finds 52 unique solutions. The distribution of machining times found by the algorithm across the starting positions can be seen in Figure 5.22. This shows that there are many local optima present in the search space for this part, offering an explanation for why the algorithms perform badly on it.

Upon further investigation of this search space it was discovered that there was an error in processing the simulations. The distributed system that was used to process every tool sequence consisted of 10 independent machines. One of the machines erroneously was not updated to use Part 4 and instead was set up to process Part 2. This meant that the final search space was a hybridisation of Part 2 and Part 4. In total, 1,808 (5%) sequences came exactly from Part 2. Comparing the results to a re-simulation of

Figure 5.22. Showing the distributions of the scores (machining times with punishment factor) obtained from the final solution returned using Steepest Ascent Hill Climbing from every starting position on Part 2-4-H.

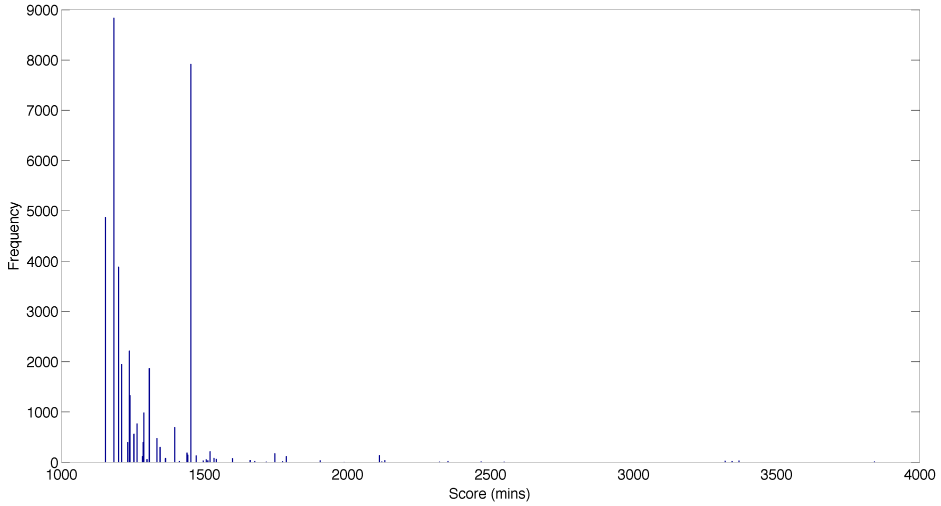
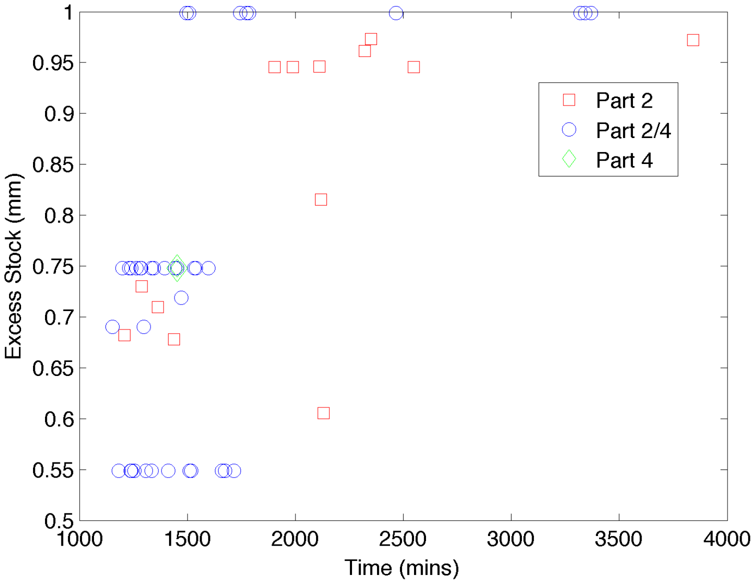


Figure 5.23. Showing the 52 local optima from Part 4's search space created by simulation error. Red squares indicate solutions that came from Part 2, blue circles are hybridisations between parts 2 and 4 and the green diamond is the solution from Part 4.



Part 4 showed that 9,662 (24%) came exactly from Part 4. The other 28,137 tool sequences (71%) appear to be a hybridisation between the two parts. This could happen by combining a cached tool sequence fragment generated for Part 2 with a new generation of a tool later in the sequence using Part 4. Out of the 52 local optima found by the SAHC algorithm, 1 (2%) came from Part 4, 13 (25%) came from Part 2 and the other 38 (73%) were hybrids. These solutions can be seen in Figure 5.23.

Part 2 is smaller than Part 4 (as seen in tables 3.1 and 3.2 from Chapter 3), which means that tool sequences working on Part 2 have a faster machining time than those that come from real Part 4 solutions. The fact that a neighbouring solution may have come from a different sized part means that there can be large differences between close solutions in discrete parts of the search space. This leads to there being many cliffs in the search space, adding to its difficulty. While this means that the actual tool sequences are not industrially relevant, it is not arbitrary, as the solutions have features of real search spaces. This presents an opportunity to test algorithms in a difficult search landscape.

While it is expected that SHC would perform badly on a search space with many “hills”, as it has no way of moving out of a local optimum, the results from Part 2-4-H showed that the memetic algorithm, HGA, performed worse than the GA, which could act as a warning against its use. It also shows which algorithms are more robust in the presence of noise, which could be created by using *Surrogate Approximation*, where the evaluation function is approximated using Machining Learning techniques (Jin, 2005), *Fitness Inheritance*, where solutions inherit fitness from their parents (Smith et al., 1995), or other methods to reduce the computational cost of evaluations, such as taking samples of generated surfaces, as used in (Krimpenis and Vosniakos, 2009).

5.4.4.2 Modifying Parameter Values and Stopping Conditions on Part 2-4-H

In all of the experiments above, RRSHC has used many more evaluations than the other algorithms. The GA used a population of 9 to limit its runtime. Although it found good solutions, the other algorithms often outperformed it. However, on the test search space of 2-4-H, it was less prone to terminating with suboptimal solutions than HGA or SHC. To see if it would perform better when it was run for longer and with a larger population size, the GA was applied to Part 2-4-H, with population sizes 9, 20, 30, 40, 50, 60, 70, 80, 90 and 100. These were repeated for 100 trials each for a full 50 generations. RRSHC was also rerun, this time using more restarts. Again, the algorithm was tested over 100 trials on each limit, with restart limits of 2 – 11 tested. Apart from these changes, the other parameter values remained the same.

The results of this experiment can be seen in Figure 5.24. It is clear that increasing population size and the number of restarts improves the search success for both algorithms. However, this comes at the expense of many extra evaluations. Generally, the number of optimal solutions increases with the number of restarts for RRSHC, while for the GA it peaks with a population size of 80. The best configuration of the GA finds the optimum in 62% of runs, while the best version of RRSHC achieves 65%. While this is an improvement over the configurations used in the original results, it suggests that the algorithms are still struggling with this difficult search space and different techniques may have to be used to further improve performance.

Figure 5.25(a) shows the distribution of machining times found over 100 runs for a GA with a population size of 100. It returns one of three solutions, the optimum and two local optima. The suboptimal solutions are close to the optimum but it finds it difficult to escape from these points. One problem could be diversity. Figure 5.25(b) shows an example run for the same configuration of the GA. The number of unseen solutions in the population falls from 100% to 60% after the first generation and continues to decrease until 30 generations, where it plateaus at around 10%. This could prevent the exploration needed to break away from suboptimal positions. In Figure 5.25(c), we see another example run from this configuration. Here the optimum solution is not found. The best solution found is shown in blue, the best unseen in red and the optimum in green. The best solution is found after only 3 generations. This suggests that once a good suboptimal solution is found it is difficult to

Figure 5.24. Bar charts showing in blue the median number of evaluations and in red the percentage that the optimum solution was found out of 100 runs for (a) GA and (b) RRSHC on Part 2-4-H.

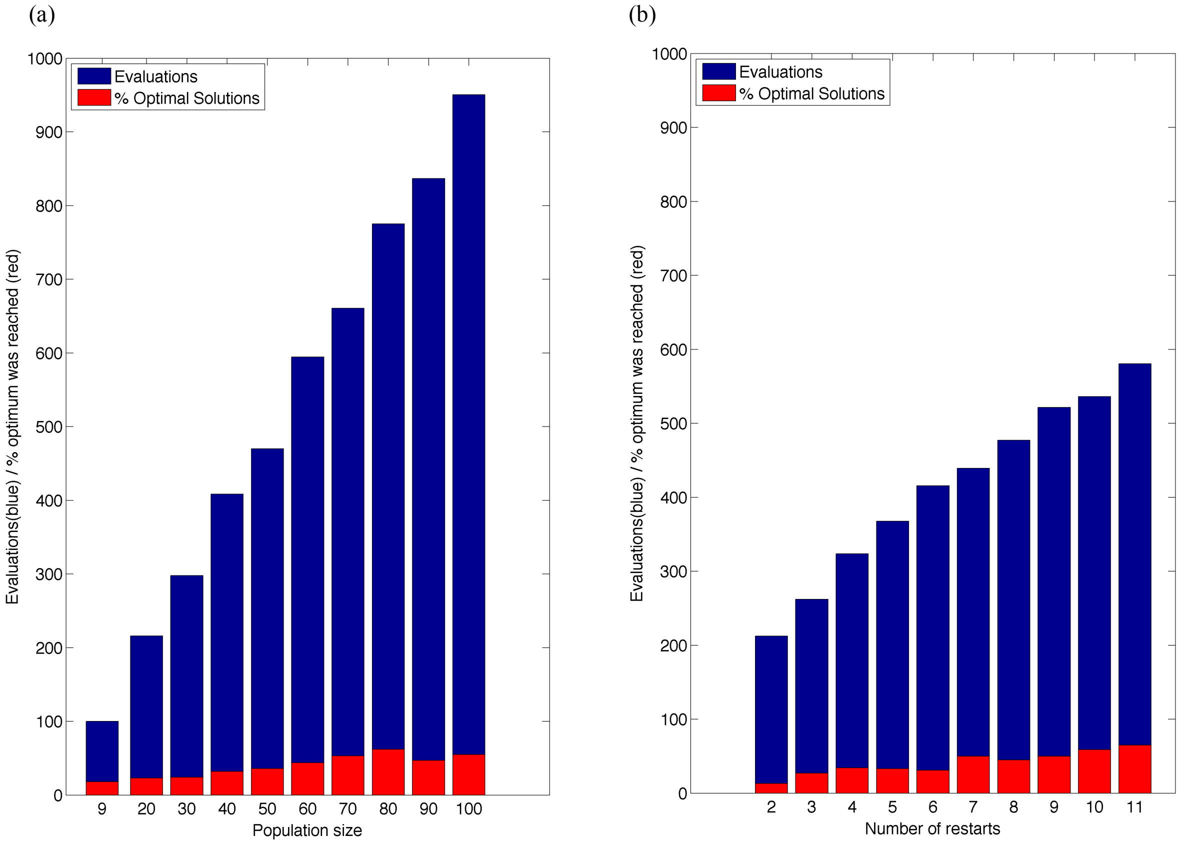
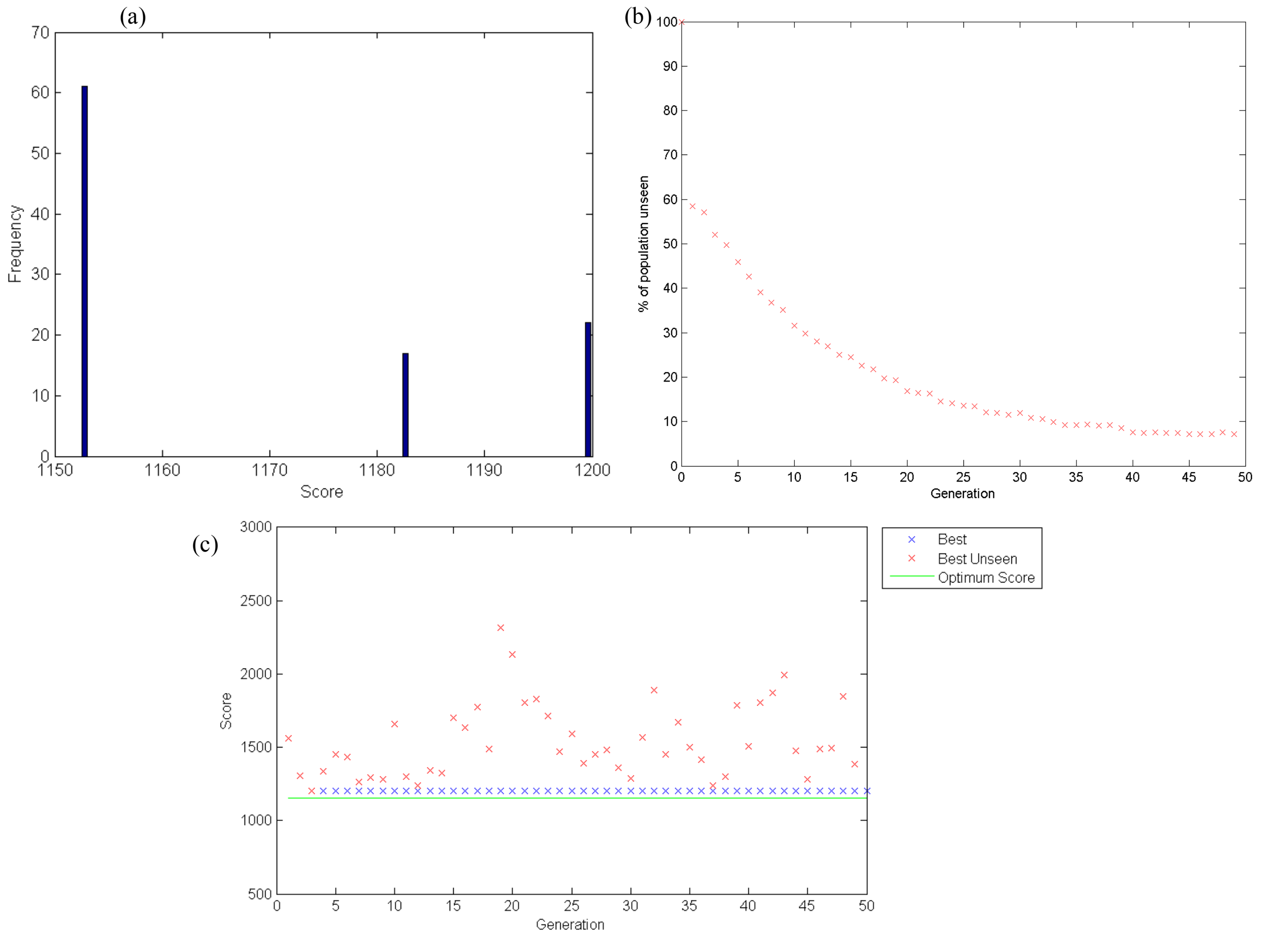


Figure 5.25. Showing for a GA with a population size of 100 on Part 2-4-H (a) a histogram for the scores achieved over 100 trials (b) the percentage of the population unique at each generation for an example run and (c) the best, best unseen and optimum solution for an unsuccessful run.



reach the optimum (which could be due to deception), as it is unable to escape this region for the next 47 generations. This could also mean that a restart strategy could be beneficial.

5.5 Comparing Evolutionary and Traditional Search on Tool Sequence Optimisation

In this thesis, we concentrate on Evolutionary Search as the main method for optimising tool sequences and tuning parameters. Using Evolutionary Search offers several advantages. Firstly, it provides an approximation search method, where good solutions can be found after only evaluating a small number. Secondly, it supports the simultaneous optimisation of continuous and discrete variables. This would be very difficult to achieve using traditional search methods such as discrete tree-based search, or continuous gradient-based methods. Thirdly, it supports the simultaneous optimisation of multiple-objectives, such as time and cost, again within the domain of combining discrete and continuous variables. However, one disadvantage of this approach is that there is no guarantee that the optimal solution will be obtained, as the algorithm can get trapped in local optima. Below, we will compare the performance of Evolutionary Search to more traditional search techniques. We compare the results obtained in this chapter to the simpler Random Search Algorithm (RSA), to see if an advantage can be expected by using a more complex stochastic algorithm. We also look at exhaustive evaluation in the form of a tree search, with and without pruning, and assess its computational cost in terms of the expected number of evaluations.

5.5.1 Comparing Evolutionary Algorithms with Random Search

For Tool Sequence Optimisation, the evaluation of tool sequences through simulation is the key determinate of computational cost. We will first compare the results seen in this chapter to Random Search. Unlike methods such as Hill Climbing or Genetic Algorithms, Random Search, where we randomly generate a specified number of unique solutions, cannot get stuck in local optima or fall into deceptive traps. However, it does not make use of any structural information from the search space, with each new decision made independently of the previous. Given this, and a uniform random distribution for solution selection, we would expect the distribution of solutions found by a Random Search Algorithm (RSA) to match the distribution of solutions in the search space. For example, Figure 4.12 in Chapter 4 shows the distribution of machining times for all unique solutions on Part 1, given the constraints discussed in that chapter. Using an RSA we would expect that the majority of solutions found would lie in the 200 – 300 minute region, while a smaller number of solutions would be found in the 100 – 200 minute and 400 – 600 minute regions. Given evaluation limits of 100, 250, 500 and 1000, Table 5.9 below shows the probabilities that an RSA will find the optimum or a high quality solution on Part 1.

This shows that running RSA with a limit of 1,000 evaluations, which would take approximately 8 hours of computational time for Part 1, we would expect to find one of the top 10 solutions 2% and the optimum solution 0.25% of the time. Comparatively, we see in the results presented above that using a Hybrid Genetic Algorithm we find the optimum solution 90% of the time and a solution in the top 10

Table 5.9. The probability and time cost in minutes of finding the optimal solution, a solution in the top 10, and a solution in the top 100 for Part 1, with various evaluation limits.

Evaluations	Optimum	Top 10	Top 100	Time Cost
100	0.0025	0.0249	0.22	50
250	0.0063	0.061	0.47	125
500	0.0126	0.119	0.72	250
1000	0.0252	0.226	0.92	500

every time, given only 50 minutes of computational time, a speedup factor of almost 10. This shows that although both are stochastic, the metaheuristic evolutionary approach greatly outperforms RSA.

5.5.2 Comparing Evolutionary Algorithms with Tree-based Search

RSA can be considered an exhaustive search algorithm. Given enough time and a memory storing previously found solutions, RSA is guaranteed to find the global optimum but follows no search path. Using traditional tree-based search methods, which incrementally build solutions, could prove more effective than random-search, if shorter solutions are of higher quality. While RSA is randomly darting around the search space, tree-based methods are methodically exploring the local search landscape.

Given the heuristics outlined in Chapter 4, Table 5.10(a) presents the number of tool sequences in the search space, separated by sequence length. The parts investigated in this chapter have an optimal sequence of either two or three tools in length. An exhaustive search would require 203 and 1,535 evaluations respectively to search through all of the two and three tool sequences. For Part 1, to search through all two tool sequences would take approximately 1.75 hours to complete compared to 0.7 for the 87 evaluations used on average by HGA (seen in the results earlier in this chapter). For Part 3, with a three tool optimal sequence, 1,535 evaluations are needed for an exhaustive search, compared to 114 required by the HGA, around 30 hours of processing time compared to 1 hour. Clearly, as the size of the tool library increases, the number of potential sequences also increases and an exhaustive search will require a larger number of evaluations. For example, in Chapter 9 we will see that with an enlarged tool library containing 26 tools (8 more than used here), the total number of sequences up to 5 tools in length adds up to 220,000. The optimal solution for Part 4 with this library contains four tools, and an exhaustive search of sequences up to this length would require 34,000 evaluations, which would take months for a single CPU to complete, given simulation times in the range of minutes.

There is however the opportunity to prune the search space by using a Branch and Bound technique (Russell and Norvig, 2003). Here, a solution is discarded if its lower bound is higher than the upper bound of another. For example, adding an extra tool in the sequence can only ever increase the machining time, so there is no advantage in expanding a solution once the excess stock limit has been reached. Here, we explore an exhaustive search through the Breadth First Search (BFS) tree-search method (Russell and Norvig, 2003), which incrementally builds sequences. The first level of the tree contains single tools ordered as nodes from the smallest on the left, to the largest on the right. All unvisited nodes are evaluated, again from left to right, before expanding the visited nodes. Nodes are expanded by adding an extra tool to the sequence, starting from the smallest on the left, restricted by the rules described in Chapter 4.

Table 5.10. Showing (a) the number of evaluations required to perform an exhaustive evaluation of all tool sequences for all parts in this chapter, (b) the number of evaluations required to perform an exhaustive evaluation of every viable solution on Part 1 using a bounding method and (c) the same for Part 3.

(a)

Length	Evals	Cumulative Evals
1	18	18
2	185	203
3	1332	1535
4	7236	8771
5	30836	39607

(b)

Length	Evals	Cumulative Evals
1	18	18
2	168	186
3	214	400
4	7	407
5	0	407

(c)

Length	Evals	Cumulative Evals
1	18	18
2	176	194
3	610	804
4	1699	2503
5	2681	5184

Table 5.11. The number of evaluations required to reach the optimum solution using a Breadth First Search with and without pruning.

Part	Evaluations to reach optimum (no pruning)	Evaluations to reach optimum (pruning)
1	55	55
2	55	55
3	1007	598
4	1002	540

The bounding method effectively discards any child sequence whose parent incurs a greater time cost than the current best-found solution meeting stock constraints, by not expanding the parent. Table 5.10(b) shows that for Part 1, an exhaustive search using this bounding method requires 186 evaluations to survey all viable solutions up to two tools in length, 400 for all three-tool, and 407 for four and five. This means that the algorithm is able to prune the search space to such an extent that it only needs to survey 1% of solutions to evaluate every potentially optimal solution.

Table 5.10(c) shows that much less pruning is possible on Part 3, which, to state again, has a three tool optimal sequence. Similarly to Part 1, very few two-tool sequences can be pruned. However, to evaluate all viable three-tool sequences 804 evaluations are needed. This is a saving of 731 evaluations but a huge increase on the evaluations required for searching on Part 1. 5,184 evaluations are required to survey the whole search space, which amounts to 13%.

On Part 1 the HGA requires 87 evaluations (87% success rate) while RRSHC requires 141 (99%). On Part 3 the HGA requires 114 evaluations (100% success rate) while RRSHC requires 168 (99%). Assuming a processing time of 30 seconds for Part 1 and 70 seconds for Part 3, and a maximum search depth of 3-tool sequences, the HGA would require 44 minutes while the exhaustive method would require 200 minutes (saving 156 minutes) on Part 1. On Part 3 the HGA search would take 133 minutes while BFS with pruning would take 938 minutes, a saving of 805 minutes (13 hours). If the entire search space up to sequences of 5 tools in length were to be considered, the exhaustive method would take 203 minutes for Part 1 and 6,048 minutes (100 hours) to complete on Part 3. The 3,403 evaluations that are required to exhaustively evaluate Part 4, which is larger and has an average simulation time of 240 seconds, would require 226 hours to complete. This suggests that although huge savings can be made using pruning, exhaustive methods can still take an extremely long time to survey the search space.

Table 5.11 shows the number of evaluations required to reach the optimum solution for parts 1 to 4 using BFS and BFS with Branch and Bound pruning. Here we see that on parts 1 and 2, which have two-tool optimal sequences, the optimum can be reached after only 55 evaluations. However, with parts 3 and 4, which have three-tool optimal sequences, using pruning 598 and 540 evaluations are needed. This gives an indication of the number of evaluations that might be needed to reach the optimum solution if we were to build up tool sequences incrementally and run for a fixed number of evaluations. Of course, we would not know that we have reached the optimal solution without running the full exhaustive search. The table shows that the number of evaluations needed clearly depends on the length of the optimal tool sequence, which is impossible to know beforehand.

Approximate methods can offer large time savings compared to exhaustive methods, with the caveat that there is no guarantee that the optimal solution will be found. These savings become greater as the search space increases, and especially if optimal sequences are long. Nonetheless, given short simulation times, small tool libraries and small depth limits, tree-based search can find optimal solutions through an exhaustive search in a relatively short amount of computational time, and may be more attractive than approximate methods, as they cannot get stuck in local optima. As tool libraries grow and longer tool sequences are required, the exhaustive approach may not be computationally viable, especially if simulation times are high. It is at this point that the approximate methods will be more desirable. For example, on Part 3 with the library used in this chapter, the HGA returns the optimal solution within 114 evaluations on average, while BFS with pruning requires a minimum of 598, without knowing when to stop looking. Given the best possible case of setting the evaluation limit to the minimum required, the HGA would still complete its search around 10 hours quicker.

The evolutionary methods considered in this thesis offer more, however, than simply time savings. The population-based strategies naturally offer multiple solutions, including those that provide trade-offs between conflicting objectives, which is explored in Chapter 7. While evaluating every possible solution could of course support this, it would inherently mean that far fewer solutions could be pruned. Additionally, traditional techniques such as tree search are not naturally extensible to input vectors that combine discrete and continuous components, an optimisation problem that is explored in Chapter 8. It should also be noted that tree-based and other traditional search methods can be integrated into a GA as a memetic algorithm, and by evaluating different methods and discovering successful search strategies, a new and improved HGA could be created.

5.6 Discussion

The results from the experiments in this chapter have shown that metaheuristic approaches work well in this problem space, and a comprehensive comparison in the performance between different algorithms has been given. This includes a methodical comparison based on both solution quality and evaluation costs, which has hitherto been lacking in the literature.

A new representation scheme has been introduced. This differs from previous work, such as (Ahmad et al., 2010; Chen and Fu, 2011), and allows variable length sequences with free ordering, supporting flat end mill, ballnose and toroidal tools. Every one of the optimal solutions included a toroidal tool, and the best solution for the test search space in Part 2-4-H used a ballnose tool at the end. This implies that supporting multiple tool types can lead to more efficient results and that their inclusion is important. This is the first time that a Tool Selection Optimisation system has included the three main cutter types.

While it has not been investigated in the literature, we have shown in the results that the choice of mutation operator contributes considerably to search success on this problem. Here, two mutation operators were experimented with and a novel gradual method was found to significantly increase solution quality and decrease evaluation costs for all of the tested algorithms.

There is no clear winner among the metaheuristic algorithms in terms of satisfying the competing demands of finding (near) optimal solutions and keeping unique evaluation numbers low. For three out of the four components tested, RRSHC found the optimum solution most often (beaten by less than half a percent on Part 3 by HGA) and for all of the parts SHC used the lowest median evaluations. For three of the four parts SHC can be said to perform better than the GA because it finds the optimum more frequently, with fewer evaluations. However, the GA greatly outperformed it on Part 2-4-H, the hybrid search space with many local optima. The HGA could be considered to perform better than SHC because for the majority of parts it found the best solution more often without a considerable increase in evaluations. However, these extra evaluations could be expensive and undesirable if a large part needs to be simulated. Similarly, the HGA may be considered superior to RRSHC because although it does not find the best solution as often, it uses considerably fewer evaluations. If a small part is used, the simulations take far less time and so evaluations are not as expensive. In this situation the evaluations criteria becomes less important and RRSHC may be seen as being the superior solution. Also, it may be very important to find the best solution available, in which case it would be better to use RRSHC, even with the likely extra evaluations. This is especially the case if the same component is going to be machined multiple times.

By analysing the search space of the different parts using the deterministic Steepest Ascent Hill Climbing algorithm, we get a good indication of where SHC outperforms the GA. On parts 2 and 3 there are two hills but it is very rare to fall into the suboptimal one. Here, SHC is able to find the optimal solution using only a small number of evaluations, while the GA is not. On Part 1, again there are two hills but the suboptimal hill is more dominating. In this case, the performance of the two algorithms is similar, although SHC uses fewer evaluations. On Part 2-4-H, there are multiple local optima and a distorted search landscape. Here the GA considerably outperforms SHC in terms a solution quality. This supports work found in (Prügel-Bennett, 2004; Fernando et al., 2012).

The memetic algorithm, HGA, produced interesting results. It improved on the performance of the GA on the three parts with simpler search spaces and outperformed SHC on Part 2-4-H. This suggests that local search speeds up its convergence but the population prevented it from getting as badly trapped in local optima as SHC. However, it performs worse than the GA in the presence of many local optima, which indicates that the SHC element encourages poor performance on these types of search spaces. It may be that using a smaller amount of local search or encouraging more diversity in the population improves this.

The results showed that increasing population size vastly improves the performance of the GA, although at the cost of extra evaluations. It is likely that the performance of the GA on parts 1, 2 and 3 would be improved by using larger population sizes. On Part 2-4-H, however, there seems to be a limit as to how much improvement can be expected by increasing population size alone. This implies that methods to encourage diversity may be successful, such as using fitness sharing and speciation (Goldberg, 1989) or demes (Whitley et al., 1999). Another method that could be used is adding a restart mechanism into the GA such as in (Coello Coello and Toscano, 2001). Finally, success has been shown in escaping local optima by using multi-objective techniques on single-objective problems (Knowles et al., 2001). This could have an added benefit in providing a selection of useful tool sequences. It is possible that in the presence of a larger search space, which could be created by using a larger tool library or having longer sequences, or in the presence of noise created by methods to reduce computational cost, the differences between SHC and the GA could become more pronounced, so it is important to explore methods to improve search performance.

5.7 Summary

In this chapter a framework has been set to apply metaheuristic algorithms to Tool Sequence Optimisation, supporting variable sequence lengths, free ordering and flat end mill, toroidal and ballnose tools. A novel gradual mutation operator has also been shown to improve performance and reduce evaluation costs. Four metaheuristic algorithms were tested, and all were able to find efficient tool sequences on average, on four different parts. When the search landscape is smooth, a simple Stochastic Hill Climber is shown to be able to very often find the optimal solution, evaluating a very small number of solutions. However, in the presence of a distorted search landscape with many local optima and relatively few routes to the optimum, SHC is considerably outperformed by a GA and RRSHC. The hybrid algorithm outperforms the GA, in simpler search spaces, and SHC, on the harsher search terrain, but was more prone to finding suboptimal solutions on the difficult part than the GA or RRSHC.

Increasing restarts on the RRSHC and population sizes on the GA led to an increase in search performance but at the cost of extra evaluations. None of the configurations of the algorithms could find the optimal solution on more than 70% of runs. One method that has been shown to escape local optima is “multi-objectivisation” - the use of multi-objective techniques on single objective problems (Knowles et al., 2001). As well as escaping local optima, multi-objective techniques can provide several solutions in a single search run, offering trade-offs between competing objectives. In the Tool Selection Problem as formulated in this chapter, there is no advantage in providing a tool sequence with a lower amount of stock, as long as it reaches the target tolerance. By using multi-objective techniques, the process planner can be presented with a selection of solutions that may enable them to make a more informed decision. In the next chapter, the key concepts behind multi-objective optimisation and multi-objectivisation will be introduced and discussed.

Chapter 6

Multi-Objective Evolutionary Algorithms and Guided Elitism

In this chapter we explore Multi-objective Optimisation and its integration into evolutionary algorithms. There are many ways that this could be useful for machining problems, and the concepts introduced throughout this chapter are applied to multi-objective implementations of the Tool Selection Problem in Chapters 7, 8 and 9. We begin with an introduction to Multi-objective Optimisation and Pareto dominance. We next describe several ways in which Multi-objective Optimisation has been implemented. The popular NSGA-II algorithm (Deb et al., 2002) is then discussed in depth, as are methods for providing preferences in multi-objective search. Experiments comparing NSGA-II and two methods for accomplishing preferential search are then presented. In Section 6.7, *multi-objectivisation*, the use of multi-objective techniques on single-objective problems is discussed and a novel method for preferential search, *Guided Elitism*, is presented. In Section 6.7.3, NSGA-II and Guided Elitism are compared to a single-objective Genetic Algorithm on the Knapsack Problem, and the multi-objectivisation approach is shown to be successful. The Guided Elitism adaptation to NSGA-II appears in (Churchill et al., 2013a).

6.1 Introduction

In the real world, many optimisation problems contain multiple objectives. For example, there are many situations where it is desirable to maximise performance and minimise cost. Imagine that we need to manufacture a product and we have several machines that we can hire. Each machine operates at a different speed and has a different rental cost but faster machines are more expensive to use. The fastest solution would utilise the quickest and most expensive machine, while the cheapest would rely on the slowest.

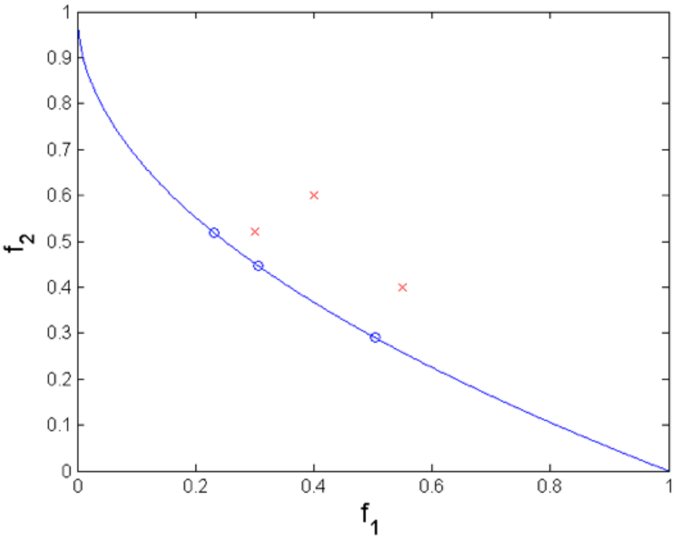
From an outside perspective, it is impossible to discern which of these solutions is better. There are two objectives, time and cost, and both solutions maximise one of the objectives while minimising the other. Where x represents our choice of machine, these objectives can be written as,

$f_1(x) = \ total\ machining\ time$

$f_2(x) = \ total\ cost$

Our task in this example is to minimise both $f_1(x)$ and $f_2(x)$. A single objective version of the problem could be to minimise just the machining time, $f_1(x)$. In this case the minimum point of the objective function would correspond to our globally optimal solution. In the multi-objective case, unless the solution that corresponds to the minimum point in $f_1(x)$ also corresponds to the minimum point for $f_2(x)$, there is no single globally optimum solution. What we have is a set of optimal solutions where

Figure 6.1. An example of a multi-objective minimisation task for 2 criteria, f_1 and f_2. Here, every cross is dominated by at least one circled solution. The blue line represents the Pareto front. Any solution on the Pareto front is non-dominated.



there is a trade off between a gain in one objective and a loss in another. Looking at the set of possible solutions the decision maker can choose the one that best represents their exact requirements at that point in time.

6.2 Multi-objective Optimisation

In this section we will introduce the key concept of Pareto dominance and methods for evaluating Multi-objective Optimisation algorithms.

6.2.1 Pareto Dominance

The concept of Pareto Efficiency is often used in a multi-objective context and is based on the 19th century work of economist Vilfredo Pareto. A solution set is considered Pareto optimal if no possible change to the decision variables of a solution exists that will provide an improvement in any one objective function without deterioration in the values of any of the others (Coello Coello et al., 2007). One solution is said to dominate another if the value of at least one objective function is superior to the other solution's and the values of none of the other objectives are worse. It is said to weakly dominate if it is not superior in every objective function. More formally, in the presence of m objective functions, for the decision vectors $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\boldsymbol{a} \prec \boldsymbol{b} \; (\boldsymbol{a} \; dominates \; \boldsymbol{b}) \qquad iff \; f_i(\boldsymbol{a}) < f_i(\boldsymbol{b}) \; \forall \; i \in [1, \ldots, m] \qquad (6.1)$$

$$\boldsymbol{a} \preccurlyeq \boldsymbol{b} \; (\boldsymbol{a} \; weakly \; dominates \; \boldsymbol{b}) \quad iff \; f_i(\boldsymbol{a}) \leq f_i(\boldsymbol{b}) \; \forall \; i \in [1, \ldots, m] \; \wedge \\ \exists \; i \in [1, \ldots, m] \; f_i(\boldsymbol{a}) < f_i(\boldsymbol{b}) \qquad (6.2)$$

Figure 6.1 shows an example in objective space of a multi-objective problem with two objective functions, $f_1()$ and $f_2()$. The blue curve shows the Pareto optimal front. Any solution on the Pareto front is non-dominated, meaning that it cannot be dominated by another solution. Circles represent solutions that are non-dominated, while crosses show dominated solutions.

In the Pareto model, all non-dominated solutions are considered to be of equal quality (Deb, 2001). Moving back to our manufacturing example, both the 'cheapest but slowest' and 'fastest but most expensive' solutions are Pareto optimal. It is up to the decision makers to choose a solution that represents a good trade off between objectives, based on their personal preferences.

For example, given non-dominated solutions with the following pairs of objective values:

(£100,300 minutes)

(£200,150 minutes)

(£300, 140 minutes)

(£500, 90 minutes)

decision makers may choose (£200, 150 minutes) because they feel it represents the best value. However, they may have a very short time frame and decide that the (£500, 90 minutes) solution is the best. However, there is no point picking a dominated solution such as (£500, 100 minutes).

6.2.2 Evaluating the Performance of Multi-objective Algorithms

In general, there are two goals in Multi-objective Optimisation, (1) to find solutions on the Pareto optimal front, and (2) to return solutions evenly spread along the front (Deb, 2001). Both of these goals need to be considered when evaluating the performance of a multi-objective algorithm.

A common method for quantifying our first goal, the quality of solutions, is to calculate the Generational Distance (Van Veldhuizen and Lamont, 2000; Deb, 2001; Coello Coello et al., 2007). This calculates the average distance between the solutions discovered and the actual Pareto front. This is achieved using the following formula,

$$GD = \frac{\sqrt{\sum_{i=1}^{n} k_i^2}}{n} \qquad (6.3)$$

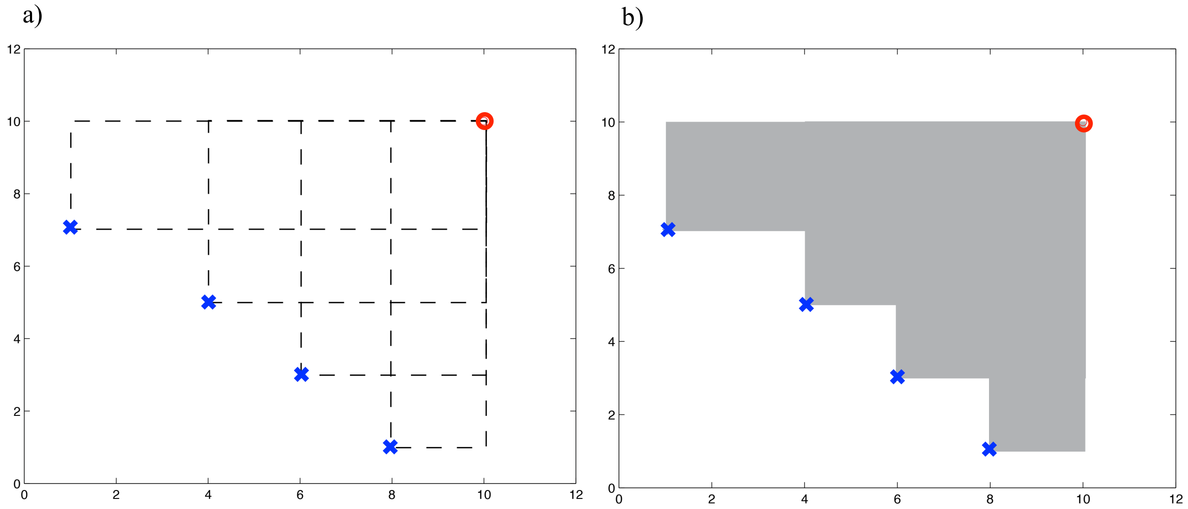
where n is the size of the members in the discovered Pareto front and k_i is the distance between the i^{th} member of the discovered Pareto front and the closest member of the true Pareto front. This requires the true Pareto front to be known, which can be impossible in real world situations.

Efficient Set Spacing (ESS) is a method used to determine the spread of solutions in the discovered Pareto front. This is calculated using,

$$S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(d_i - \bar{d}\right)^2} \qquad (6.4)$$

where d_i is minimum of the sum of the absolute difference in objective function values between solution i and any other solution in the non-dominated set, and $\bar{d}$ is the mean of these values (Deb, 2001).

Figure 6.2. Plots showing for a bi-objective problem (a) the hyperrectangles and (b) the union of the hypervolume for four points (blue crosses) from a reference point of (10,10), circled in red.



6.2.2.1 Hypervolume Metric

In contrast to the above, a more commonly used method for evaluation is the hypervolume metric, which provides a single value to represent a measure of spread and diversity (Ziztler and Thiele, 1998; Deb, 2001; Coello Coello et al., 2007). Using a reference point, which can be a vector of the worst possible objective function values, the hyperrectangle between each member of the discovered Pareto front and this reference point is calculated. The volume covered by the union of these hyperrectangles is then calculated, giving the total hypervolume covered by the Pareto front from the reference point.

This is illustrated in Figure 6.2 for a bi-objective problem. Here we have the discovered Pareto front, P_f, consisting of the points:

(1,7); (4,5); (6,3); (8,1)

and the reference point (10,10). Figure 6.2(a) shows the hyperrectangles for these points, which are simply rectangles because of the 2D objective space. Figure 6.2(b) shows the total hypervolume for these hyperrectangles, which is the union of the areas of the individual hyperrectangles. The total hypervolume is,

$$H_V = ((10-8) \text{ x } (10-1)) + ((8-6) \text{ x } (10-3)) + ((6-4) \text{ x } (10-5))$$
$$+ ((4-1) \text{ x } (10-7)) = 51$$

If we remove the point (6,3), the hypervolume becomes,

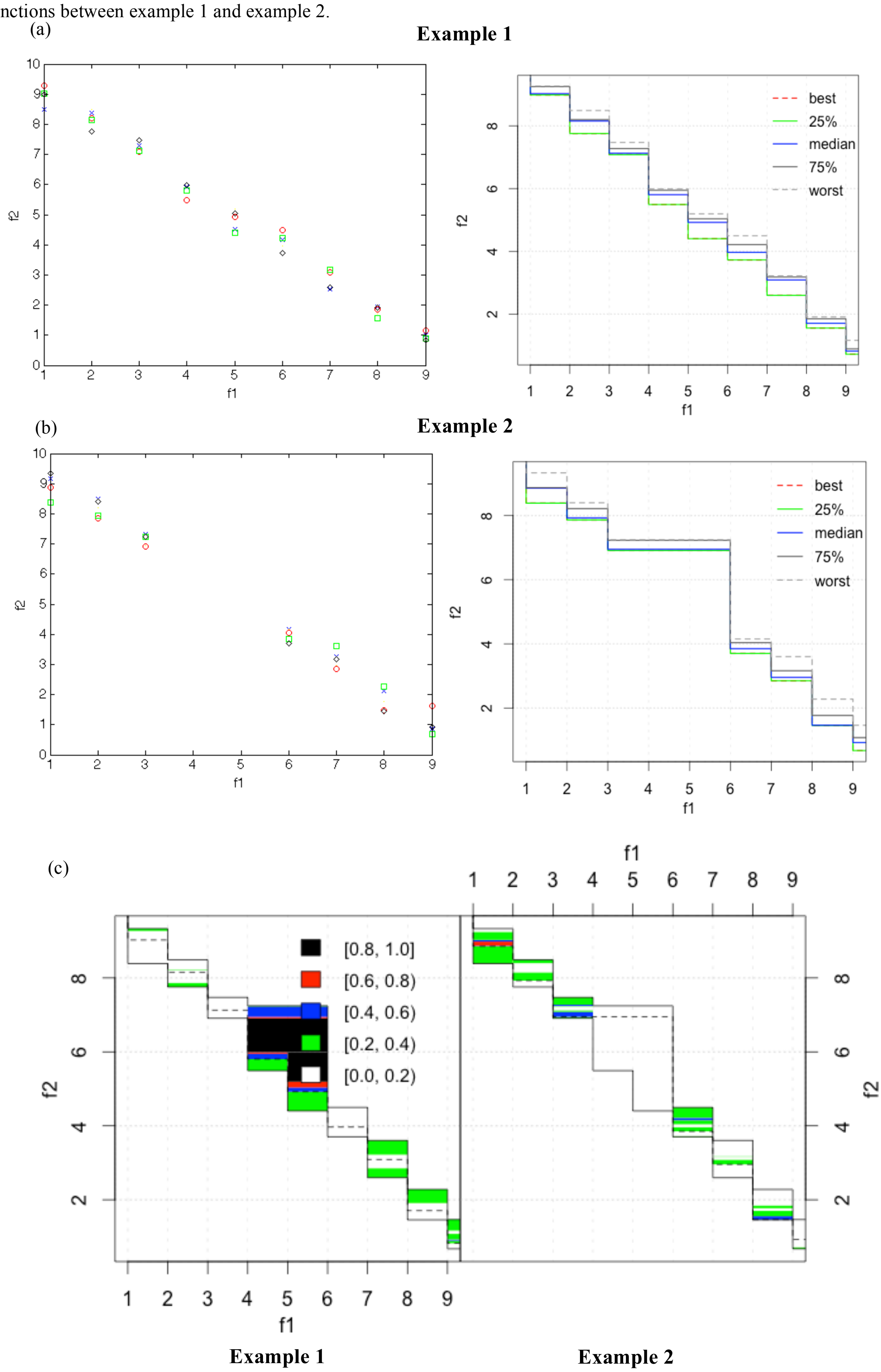
$$H_V = ((10-8) \text{ x } (10-1)) + ((8-4) \text{ x } (10-5)) +$$
$$+ ((4-1) \text{ x } (10-7)) = 47$$

This shows that finding a better spread of non-dominated points will lead to a larger hypervolume.

6.2.2.2 Empirical Attainment Function

The performance measures described above assign value to how much closer one set of non-dominated solutions is to the Pareto front than another, or how much further the solutions are spread along the front. However, no information is returned about where solutions commonly occur in objective space.

Figure 6.3. Showing (a) all of the points obtained over 5 trials (left) and the surfaces attained in the best, 25%, median, 75% and worst case (right) for example 1; (b) all of the points obtained over 5 trials (left) and the surfaces attained in the best, 25%, median, 75% and worst case (right) for example 2; and (c) the differences in Empirical Attainment Functions between example 1 and example 2.



The attainment function can be used for this purpose. For each point in objective space, a probability is assigned to whether that point will be attained, i.e. dominated or equalled, by sets of points (López-Ibáñez et al., 2010). Given the output of several independent trials of a multi-objective algorithm, the attainment function can be empirically derived. An attainment surface shows the boundary of non-dominated points attained in x% of cases.

In Figure 6.3 (a) and (b) we see scatter plots and attainment surfaces for the outputs of two example multi-objective algorithms over five trials. The algorithm labelled *example 2*, is missing a middle part of the Pareto front. The differences in the surfaces attained by each algorithm can be plotted by calculating the attainment function for all of the points found by both algorithms, and then calculating how many times one algorithm attained that point over the other (López-Ibáñez et al., 2010). Figure 6.3 shows a plot of the positive differences between the attainments of *example 1* (on the left) and *example 2* (on the right). We can see on the left plot that the error shaded in black is the area of the front missing in *example 2*.

6.3 Evolutionary Multi-objective Optimisation

The standard way that Multi-objective Optimisation has been applied to evolutionary algorithms is through the use of Pareto dominance for comparing solutions (Coello Coello et al., 2007). In this section we are going to explain how this was arrived at by beginning with weighted aggregation and VEGA, and then moving to Pareto-based methods. Recent work has shown that comparing solutions using Pareto dominance alone does not handle many-objectives (four or more) very well, and although many-objective optimisation is not used in the work in this thesis, it is discussed here for completeness.

6.3.1 Weighted Aggregation

In this approach, an aggregation function is used which sums the weighted values from a number of objective functions. The weighted aggregation function for *m* objectives can be represented as:

$$f_w(x) = \sum_{i=1}^{m} w_i f_i(x) \qquad (6.5)$$

where w_i is the weight on i^{th} objective function. The weights are set based on the decision maker's preference. In the manufacturing example, we may decide that time is worth twice as much as cost, so our weighted function will be:

$$f_w(x) = total\ machining\ time + 0.5(total\ cost)$$

Returning to our earlier example solutions, we evaluate them as:

$$f_w(£100, 300\text{ minutes}) = 350$$

$$f_w(£200, 150\text{ minutes}) = 250$$

$$f_w(£300, 140\text{ minutes}) = 290$$

$$f_w(£500, 90\text{ minutes}) = 340$$

This makes the optimum solution (£200,150 minutes) using this weighting scheme. Weighted aggregation is very often used in practice. One of its key advantages is that it allows for existing single-objective techniques to be used. However, for many problems it can be difficult to set the weighting value before running the optimisation, or priorities may change at a later date, which can mean that multiple runs are needed. Multiple runs also will be needed to approximate the Pareto front (Ziztler, 1999). Coello Coello et al. write that using weightings can prevent optimal solutions from being found when the Pareto front is non-convex, although this can be circumvented by using a non-linear weighting function (2007).

6.3.2 VEGA

Schaffer's Vector Evaluated Genetic Algorithm (VEGA) is the earliest and most commonly used population-based approach for Multi-objective Optimisation (1985). It is characterised by switching objective functions. At each generation the population is randomly divided into subpopulations, which are evaluated using a particular objective function. Selection is performed across the population meaning that individuals evaluated with different objective functions are mated together. This can produce a population of solutions that perform well on the different objectives. Pareto dominance is not used to compare solutions. This creates a limitation because it means that solutions on the Pareto front that represent a good balance between the different objectives are often filtered out of the population. This occurs due to these solutions not performing highly on an individual objective, so in a given generation they may not be selected (Coello Coello et al., 2007).

6.3.3 Pareto-based Approach

In the Evolutionary Multi-objective Optimisation (EMO) community, Pareto-based approaches have been the subject of a great deal of research. Goldberg first suggested using Pareto dominance to select individuals (Goldberg, 1989). In this approach individuals are not selected based on how they fare on individual or aggregated objectives but whether or not another individual dominates them. The goal in Pareto-based methods is to return a diverse spread of Pareto optimal solutions. Commonly, solutions are compared using a Pareto-ranking method. In (Fonseca and Fleming, 1993) solutions are ranked by the number of solutions in the population that they are dominated by. In (Srinivas and Deb, 1994), solutions are ranked using non-dominated sorting, which assigns solutions into hierarchical Pareto fronts.

A key factor for the successful performance of Pareto-based algorithms is diversity preservation. By selecting based only on dominance it is easy for algorithms to converge to certain sections of the Pareto front. Popular methods for preserving and promoting diversity have been niching and fitness sharing (Deb and Goldberg, 1989; Fonseca and Fleming, 1993; Horn et al., 1994; Deb et al., 2002).

There is a huge variety of different Pareto-based algorithms available for Multi-objective Optimisation, and a comprehensive review can be found in (Coello Coello et al., 2007). However, arguably the most popular are the Non Dominated Sorting Algorithm II (NSGA-II) (Deb et al., 2002), Strength Pareto Evolutionary Algorithm 2 (SPEA2) (Ziztler et al., 2001), and the Pareto Archived Evolution Strategy (PAES) (Knowles and Corne, 2000). In this thesis, NSGA-II is used extensively and is presented in greater detail below.

6.3.4 Limitations of the Pareto-based Approach and Many-Objective Optimisation

Pareto-based methods, i.e. those that use Pareto ranking in selection, are able to protect and select for a diverse range of solutions that offer a good trade-off between different objective functions. However, recently they have been shown to perform poorly when faced with ‘many objectives’, which refers to problems with more than three objectives (Ishibuchi et al., 2010; Hadka and Reed, 2013). This is due to a lack of selection pressure caused by almost all individuals in a population being non-dominated in high dimensional objective space, preventing a movement towards the true Pareto front.

One solution to this problem is through Indicator-based methods, which replace or augment Pareto ranking. For example in SMS-EMOA (Beume et al., 2007), solutions are first sorted by their Pareto rank. Each individual is then evaluated using the hypervolume performance indicator, described above. The individual with the worst Pareto rank and hypervolume contribution is replaced by a new solution. A downside of SMS-EMOA is that the hypervolume calculation can be computationally expensive but approximation methods, such as using achievement scalarising functions as discussed in (Ishibuchi et al., 2010) can reduce this considerably.

Another popular method has been decomposition, with Zhang and Li’s MOEA/D the most popular example (2007). Here, the multi-objective problem is decomposed into an array of scalar, single-objective functions that use different weights for each objective. Each individual in the population is assigned fitness based on a unique scalar function, resulting in each scalar function being optimised simultaneously. New solutions are created by performing genetic operators on *neighbour* solutions, based on a neighbourhood function, with neighbours defined by the distance between the scalar functions. MOEA/D can find an evenly spread distribution of solutions even with a small population size (Zhang and Li, 2007).

In (Wang et al., 2013), a co-evolutionary algorithm is presented to deal with many objectives, Preference-Inspired Co-evolutionary Algorithm (PICEA-g). Here, individuals and target values are co-evolved. Individuals are assigned fitness based on how many target values they dominate, as well as the other individuals they dominate. Target values are assigned fitness based on the number of individuals that dominate them. The use of the target values allows more comparability between solutions, resulting in more selective pressure towards the Pareto front. PICEA-g was shown to regularly outperform MOEA/D with 4 objectives or more.

6.4 Elitist Non-Dominated Sorting Genetic Algorithm II

Deb et al.’s Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) (2002) is a very popular Pareto-based algorithm, which has been used on many discrete and continuous optimisation problems. For example, in manufacturing it has been used in Machine Scheduling (Berrichi et al., 2009) and Machining Parameter Optimisation (Datta and Deb, 2009; Yang and Natarajan, 2010). NSGA-II is an enhanced version of Srinivas and Deb’s NSGA (1994). The original algorithm was based on Goldberg’s ranking method, where individuals are ranked by their dominance class (1989).

Improvements were made in the second version of the algorithm to reduce computational complexity and provide better convergence towards the Pareto front. In this section we first introduce the methods used for Pareto-ranking (Fast Non-dominated Sorting) and diversity preservation (Crowding Distance Assignment) before describing how the algorithm is implemented.

6.4.1 Fast Non-Dominated Sort

This procedure ranks each individual in a population based on Pareto dominance. It works by separating solutions into a series of non-dominated fronts. A front contains members that are all non-dominated with respect to each other. If we remove the current non-dominated front from the population, there will be previously dominated solutions that are now non-dominated. These make up the next front, and this can be repeated until all the fronts have been assigned. The Fast Non-Dominated Sort (FNS) method performsthis procedure in a computationally efficient manner, decreasing the complexity of the original NSGA from $O(n^3)$ to $O(n^2)$.

FNS works by comparing each member of the population to each other member and calculating how many members, n_k, an individual, k, is dominated by and adding each member that it dominates to the set, S_k. At the end of this loop, all solutions where n_k = 0 are in the first Pareto front, as they are not dominated by any others. All of the solutions dominated by these solutions are members of their dominated sets. For each member i of the non-dominated front, we then go through each individual, *y*, from the set S_i and subtract 1 from n_y. If at the end of this loop an individual *y* now has $n_y = 0$, it is put into the next Pareto front. We can repeat this procedure until all members have been assigned to a Pareto front. The index of the Pareto front that a member has been assigned to determines its Pareto ranking, with lower indexes superior. Pseudocode for this can be seen in Algorithm 6.1.

6.4.2 Crowding Distance Assignment

The crowding distance assignment in NSGA-II preserves and promotes diversity. If the Pareto front is larger than the number of members in the population, in a discrete problem, or if we are looking at a continuous problem, it is impossible to find the entire Pareto optimal front. In this situation we want to approximate the non-dominated front by finding solutions that are evenly spread along the front. In NSGA-II this is achieved by assigning each member, k, a crowding distance score, cd_k. This score is a measure of how crowded a point is in objective space. Initially $cd_k = 0$. After the solutions have been assigned a Pareto rank, the diversity in each front is considered separately. For each objective, the individual members in the front are sorted by their normalised objective value. The first and last members are given an infinite score. The remaining members are given a score calculated by the Taxicab distance between the normalised objective values of the solutions either side of the member in question, which is added to cd_k. An example of this procedure can be seen in Figure 6.4. Here, the crowding distance for individual i, is the distance between the f_1 value for *i -1* and *i + 1*, summed with the distance between the f_2 value for *i -1* and *i + 1*. This gives,

0.33 + 0.22 = 0.55

Pseudo-code for this procedure can be seen in Algorithm 6.2.

Algorithm 6.1: Pseudo-Code for Fast Non-Dominated Sorting

```
for each member, k, in population P
    initialise the set, $S_k$ = an empty list
    initialise the domination count, $n_k = 0$
    for each member q in population P
        if k dominates q then
            add q to $S_k$
        else if q dominates k then
            $n_k$ += 1
        end if
    end for
    if $n_k$ == 0 then
        Add k to the first Pareto front, $F_1$
    end if
end for
z = 1
while $F_z$ is not an empty set
    initialise the set $F_{z+1}$ = empty list
    for each member i in set $F_z$
        for each member y in set $S_i$
            $n_y = n_y - 1$
            if $n_y$ == 0 then
                Add y to the next Pareto front, $F_{z+1}$
            end if
        end for
    end for
    z = z + 1
end while
```

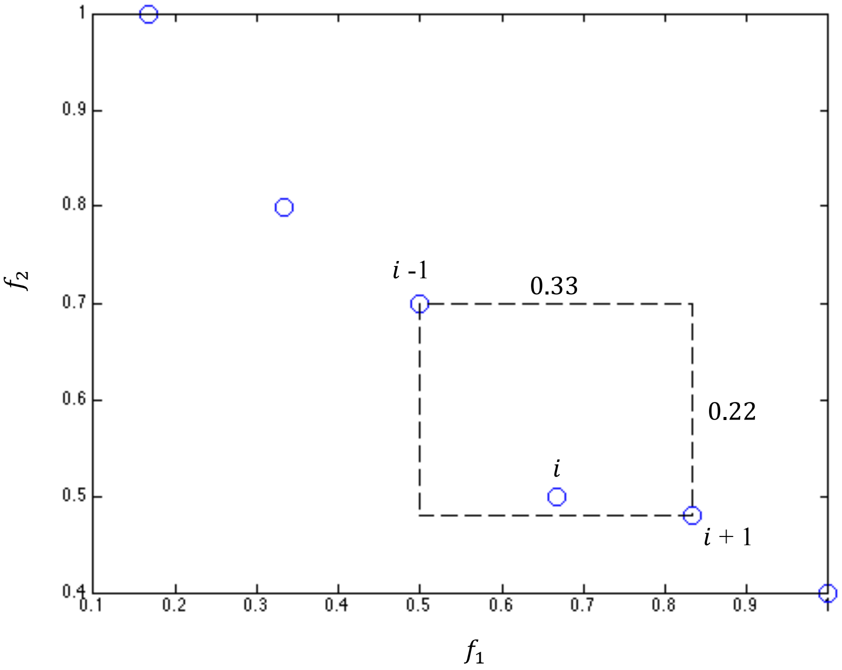
Algorithm 6.2: Pseudo-Code for the crowding distance assignment in NSGA-II

```
initialise the crowding distance to 0 for each solution
for each objective, do
    sort the solutions in the Pareto front according to their value for this objective
    normalise the objective values by dividing by the largest value
    set the crowding distance of the first and last solutions to infinity
    for solutions from solutions[start + 1] to solutions[end – 1], do
        subtract the normalized objective value to the left of the solution from
        the value to the right
    end for
end for
```

6.4.3 The Main Algorithm

The development of the NSGA-II algorithm was motivated by improving the convergence of solutions towards the true Pareto front and finding a diverse spread of solutions along the Pareto front. This is achieved through selecting solutions using Pareto ranking based on Non-Dominated Sorting and elitism (quality) and secondarily through using crowding distance (diversity). Selection occurs twice in the algorithm. First new individuals are generated by selecting two parents through binary tournament selection. An individual wins a tournament if it has a lower Pareto ranking than another. If they have

Figure 6.4. Assigning crowding distance to individual, *i*, on a bi-objective problem. The Taxi-cab distance between *i + 1* and *i -1* is calculated for f_1 and f_2 and summed, giving 0.55.



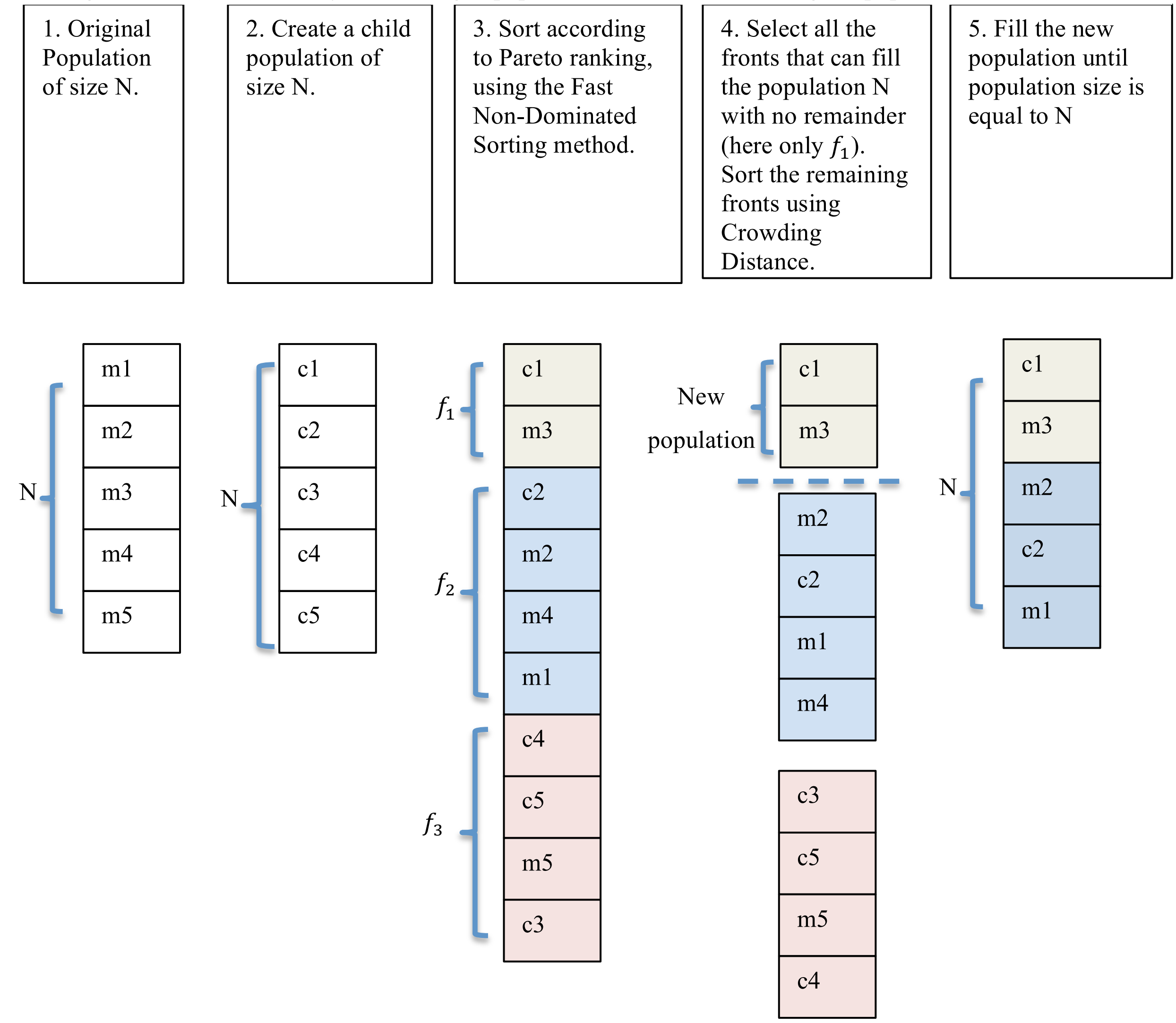
Algorithm 6.3: Pseudo-Code for NSGA-II

```
create a random population, p
while evaluations < budget, do
        p’ = ∅; new_population = ∅
        while p’ < pop_size, do
                parents = binary_tournament_selection(p)
                create new offspring through mutation and recombination
                add offspring to p’
        end while
        p = p + p’
        fast non-dominated sort population
        assign a crowding distance value to each member of the population
        sort the population using Pareto ranking and then crowding distance
        while new_population < pop_size, do
                remove top ranked members of p and add to new_population
        end while
        p = new_population
end while
```

the same Pareto ranking, i.e. they are in the same front, they are compared using crowding distance. Selection also occurs in the form of elitism. At each generation, a new population is combined with the previous one. This is then sorted according to Pareto ranking, and secondarily, crowding distance, before reducing the population to its original size. This means that in a new generation a non-dominated solution will never be replaced by a dominated solution.

The NSGA-II algorithm works in the following way. An initial population, *p*, of size, *N*, is created randomly. At each generation a new population *p*’, also of size, *N*, is created using evolutionary operators and binary tournament selection. Population *p’* is added to *p*, which is then sorted using the Fast Non-Dominated Sort method, assigning a Pareto ranking to each individual. Each individual is additionally assigned a diversity score using the Crowding Distance Assignment method. All of the

Figure 6.5. Stages in creating a new population in NSGA-II. A child population is combined with the original population to create an enlarged population (1 – 3). The enlarged population is then sorted by Pareto rank and crowding distance (3 – 4). Finally, The enlarged population is reduced to the original population size (5).



1. Original Population of size N.

2. Create a child population of size N.

3. Sort according to Pareto ranking, using the Fast Non-Dominated Sorting method.

4. Select all the fronts that can fill the population N with no remainder (here only f_1). Sort the remaining fronts using Crowding Distance.

5. Fill the new population until population size is equal to N

lowest ranked fronts that can be added to a new population in their entirety without exceeding size *N* are moved to a new population. After that, each remaining front is sorted by the crowding distance measure of its individual members. Starting from each front in order of their Pareto ranking, the members of the front are added to the new population in descending order based on their crowding distance measure, until the new population reaches size N. In this way, non-dominated solutions are never lost in favour of dominated ones, so there is elitism of the Pareto optimal front. The creation of a new population is illustrated in Figure 6.5. Pseudocode for the NSGA-II algorithm can be seen in Algorithm 6.3.

6.5 Preferences in Multi-objective Search

In the Pareto-based approach, solutions that are non-dominated are considered to be incomparable. This is because each objective is evaluated with equal importance. In real world situations, using classical Pareto-based techniques, Decision Makers (DMs) will make the choice about which solution represents the best trade off between objective values, when presented with solutions approximating the Pareto optimal front.

Using preferences to make choices at the end of search can be disadvantageous because the entire search process has been taking a neutral view and spending time and computations to provide evenly spread solutions. Concentrating on regions of the objective space of interest to the DMs could enable search to reach these areas with a higher likelihood and therefore provide more interesting solutions, to improve their solution pool. In directing the search towards preferred solutions, it can be seen as providing more exploitation to the broad exploration that takes place in Pareto-based Multi-objective search. Below, we will discuss methods that have been used to incorporate user preferences in the search space before discussing two specific techniques.

6.5.1 Methods to Incorporate Guidance into Search

Preferential search has been incorporated into Evolutionary Multi-objective Algorithms in two main ways – objective weighting and diversity modifications (Branke and Deb, 2004). Weighting methods modify the dominance relationship between solutions. This can have the effect of making previously incomparable solutions comparable, and biased towards the objective value deemed more important by the DM. It should be noted that this weighting is different to that of Aggregated Weighted Functions, which transform multi-objective problems into single-objective ones.

Diversity modification changes the way an algorithm spreads solutions along the Pareto front. It can be altered so that solutions are biased towards a particular region in objective space. Dominance weighting has mainly been achieved through weighting functions (Branke et al., 2001; Friedrich, 2011), while diversity modification has been based on using reference points in objective space (Deb and Sundar, 2006; Wickramasinghe et al., 2009). (Thiele et al., 2009) have taken an interactive approach to preferences, which can be useful if it is difficult for a DM to quantify the preference in advance.

6.5.2 Guided Multi-objective Evolutionary Algorithm

Branke et al. introduced an objective weighting approach (G-MOEA) to incorporating user preferences in Multi-objective Optimisation (Branke et al., 2001; Branke and Deb, 2004). The approach is most simply used with two objectives. The DM is asked at most how many units, a_{12}, an improvement of one unit in objective f_2 is worth in terms of degradation of objective f_1. Similarly, the DM also chooses how many units, a_{21}, of objective f_2 a one unit improvement in f_1 is worth. The dominance relationship for a minimisation task now becomes,

$$\boldsymbol{x} \prec \boldsymbol{y}\ (\boldsymbol{x}\ dominates\ \boldsymbol{y}) \qquad iff\ (f_1(\boldsymbol{x}) + a_{12}f_2(\boldsymbol{x}) \leq (f_1(\boldsymbol{y}) + a_{12}f_2(\boldsymbol{y})) \wedge$$
$$(a_{21}f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) \leq a_{21}f_1(\boldsymbol{y}) + f_2(\boldsymbol{y})) \qquad (6.6)$$

with an inequality in at least one case. The original dominance function can be replicated if,

$$a_{12} = a_{21} = 0$$

Algorithm 6.4: Pseudo-Code for the crowding distance assignment in R-NSGA-II

```
initialise the crowding distance to 0 for each solution
for each solution, do
    set the solution's crowding distance to the normalised distance to the closest reference point
end for
```

6.5.3 Reference Point NSGA-II

Deb and Sundar's Reference Point NSGA-II (R-NSGA-II), provides for preferential search by modifying NSGA-II's Crowded Distance Assignment (2006). A DM specifies reference points in *M*-dimensional objective space that represent areas of interest. While dominance rules still apply, where two similarly ranked solutions are compared, the one closest to a reference point is selected, instead of using their distance from other solutions. The normalised distance, *d*, that a solution $\boldsymbol{x}$ is from a reference point, $\boldsymbol{r}$ is determined using the following function:

$$f(\boldsymbol{x},\boldsymbol{r}) = \sqrt{\sum_{i=1}^{M} \boldsymbol{w}_i \left(\frac{f_i(\boldsymbol{x}) - \boldsymbol{r}_i}{f_i^{max} - f_i^{min}}\right)^2} \qquad (6.7)$$

where $\boldsymbol{r}_i$ is the i^{th} component of reference point $\boldsymbol{r}$, *M* is the number of objectives and (f_i^{max}, f_i^{min}) are the maximum and minimum objective values of the i^{th} objective function. A weight vector, $\boldsymbol{w}$, can be specified if a DM deems some objectives are more important than others. In order to maintain diversity, an ε parameter can be specified. If solutions are within ε of each other, they are grouped together. One randomly determined solution keeps its assigned crowding distance, while remaining solutions in the group are assigned large crowding distances to discourage selection. In the case of multiple reference points, only the closest reference point to a solution is chosen for crowding distance assignment, determined according to the function in equation 6.7 above. Pseudo-code for crowding distance assignment in R-NSGA-II can be seen in Algorithm 6.4.

6.6 Multi-objective Optimisation Examples using NSGA-II and Preferential Search

In this section, we test NSGA-II, G-MOEA and R-NSGA-II on benchmark continuous and discrete problems. These experiments are performed to provide working examples of the methods described above and to assess whether they are appropriate for work on Tool Selection problems, which have discrete properties, and Machining Parameter optimisation, which has continuous properties.

6.6.1 A Continuous Problem: ZDT1

Ziztler, Deb and Thiele introduced a suite of benchmark continuous functions, which have become the de facto standard for evaluating multi-objective algorithms (Zitzler, 2000; Coello Coello et al., 2007). To exemplify NSGA-II and preferential search extensions, we apply the algorithms to the ZDT1 function from this suite.

The goal is to find the optimum configuration of the decision vector $\boldsymbol{x}$ that will minimise the functions $f_1(\boldsymbol{x}), f_2(\boldsymbol{x})$, where,

$$\boldsymbol{x}_i \in \{\mathbb{R}\,|\,0 \le x_i \le 1\}$$

$$f_1(\boldsymbol{x}) = \boldsymbol{x}_1 \qquad (6.8)$$

$$f_2(\boldsymbol{x}) = g(\boldsymbol{x})\left(1 - \sqrt{\frac{x_1}{g(x)}}\right) \qquad (6.9)$$

$$g(\boldsymbol{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \qquad (6.10)$$

In these experiments, n, the number of decision variables in $\boldsymbol{x}$ is 30. The Pareto optimal front occurs when $g(\boldsymbol{x}) = 1$.

All of the algorithms represent solutions in the same way. An individual's chromosome is split into a string containing 30 real valued numbers between 0 and 1. At each generation, there is a 0.1 chance of recombination, where two parents are chosen by binary tournament selection. Applying a one-point crossover between the parents generates two children. There is a 1/n chance of ±0.001 being added to each of the 30 values in the string. The population size is set to 100 individuals.

6.6.2 A Discrete Problem: Unitation Versus Pairs

Horn et al. tested their multi-objective algorithm on the *unitation versus pairs* problem (1994), as did (Menczer et al., 2000). For a given bit string of length, *n*, the problem is to maximise the number of ones (objective 1) and also the number of consecutive pairs (objective 2). A consecutive pair in the bit string is defined as either a 0 occurring next to a 1, or a 1 occurring next to a 0. The two objectives are given as,

$$\boldsymbol{x}_i \in \{0,1\}$$

$$f_{ones}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i \qquad (6.11)$$

$$f_{pairs}(\boldsymbol{x}) = \sum_{i=1}^{n-1} (\boldsymbol{x}_i \oplus x_{i+1}) \qquad (6.12)$$

Figure 6.6(a) shows every possible result for a string of 10 bits. The six solutions on the Pareto front are labelled with circles. For an n-bit problem, the chromosome for an individual is simply a binary string containing *n* bits. The chromosomes are initialised randomly. There is a 0.1 chance of recombination through one-point crossover. Every individual has a *1/n* chance of each bit being flipped at each generation. Note that this is a maximisation task, and so all definitions apply with inequalities reversed.

Figure 6.6. Graph showing (a) the search space for a 10-bit *unitation versus pairs* problem. Solutions on the Pareto front are circled (b) all of solutions (blue crosses) found in an example evolutionary run.

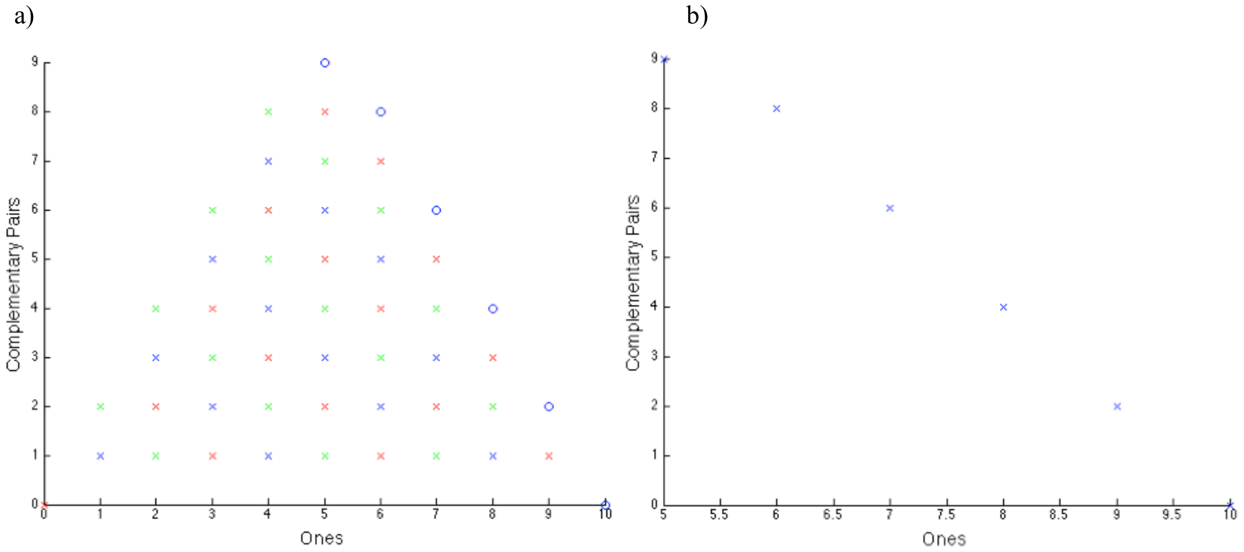
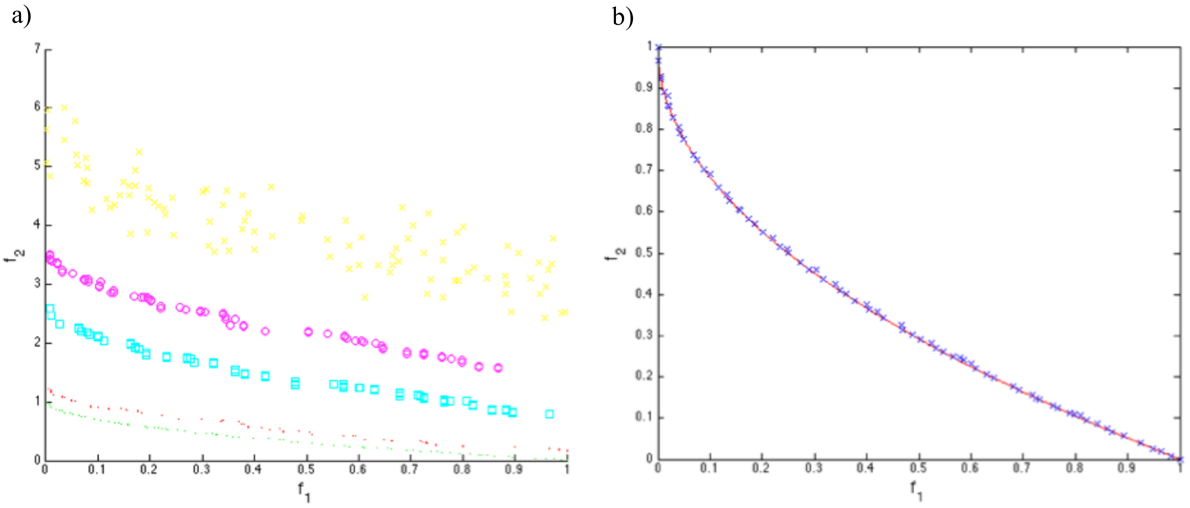


Figure 6.7. Showing (a) results of NSGA-II in objective space at various generations, when minimising ZDT1. Yellow solutions occur at 0 generations, magenta at 50 generations, cyan at 100, red at 200 and green at 300 (b) the Pareto optimal front (red line) and results from NSGA-II (blue crosses) after 500 generations.



6.6.3 Results

6.6.3.1 NSGA-II

For ZDT1, Figure 6.7 shows the results of NSGA-II at various stages in the evolutionary cycle during a single evolutionary run. In Figure 6.7(a), we can see that there is a clear improvement as the number of generations increase. After 200 generations we start to get close to the Pareto optimal front. Figure 6.7(b) shows the algorithm after 500 generations. We can see that the solutions are very close to the Pareto front, which is shown with a red line. The solutions are also well spaced out across this line, meaning that they represent a good approximation of the front.

Figure 6.6(b) shows an example run from the 10-bit *unitation versus pairs* problem. The NSGA-II algorithm is able to find all six solutions on the Pareto front. A population size of 15 was used and ran for 20 generations.

6.6.3.2 G-MOEA

A version of NSGA-II using G-MOEA objective weighting was applied to ZDT1. A population of 100 was applied for 500 generations. 121 different weighting configurations were tested, varying a_{12} and a_{21} from 0.0 to 1.0 in increments of 0.1. Figure 6.8 shows the results for these different values. We can see that G-MOEA is able to successfully guide the search towards different regions of the search space by altering the a_{12} and a_{21}. Solutions can be concentrated towards better f_2(x) values, i.e. concentrated on the top of the y-axis. This is achieved by keeping a_{21} at 0 and increasing a_{12}. Conversely, solutions can also be concentrated towards $f_1(x)$, i.e. towards the right of the x-axis, by keeping a_{12} at 0 and increasing a_{21}. Search can also be concentrated on the middle region. For example, $a_{12} = 0.7$ and $a_{21} = 1.0$, concentrates solutions in a narrow band in the middle of the Pareto front, while $a_{12} = 0.6$ and $a_{21} = 0.9$ concentrates on a wider middle band. One potential challenge with this approach is that it may be difficult to choose the exact weightings a priori, if a specific region is desired.

Figure 6.9 shows NSGA-II with G-MOEA applied to a *unitation versus pairs* problem, again varying values from 0.0 to 1.0 for a_{12} and a_{21} in increments of 0.1. Differing to the NSGA-II experiment above, this time a 30-bit problem is used, in order to expand the size of the Pareto front and provide a better assessment of preferences. A population size of 10 is evolved for 500 generations. Changing the values of the weighting has a clear effect on the region of the search space explored. However, on this discrete task it appears to be much less clear how to tune the values to pinpoint the exact region of space desired. For example, keeping a_{12} at 0 and increasing a_{21} skews the search towards $f_2(x)$, i.e towards values at the top of the y-axis. There does not appear to be a combination of the presented values that can achieve the same for $f_1(x)$, returning solutions at the bottom of the x-axis. Unlike with ZDT1, it seems difficult to concentrate solutions towards narrow bands of solutions in the middle of the front. It may be that the weightings encourage search to fall into local optima. Comparing this to Figure 6.8, it appears that gradually changing in the weights G-MOEA is much less smooth and predictable on this discrete problem than on the continuous function and it may not be possible to transfer weight rules from one problem to another.

Figure 6.8. The effect of applying different weights to objectives on G-MOEA, when optimising the ZDT1 function. For each subplot, $f_1(x)$ is on the x-axis and $f_2(x)$ is on the y-axis. Blue crosses show solutions, and the red curve shows the Pareto front. Weight a_{21} increases from 0.0 to 1.0 in increments of 0.1, from subplots from the left to the right. Weight a_{12} increases from 0.0 to 1.0 in increments of 0.1, from subplots from the top to the bottom

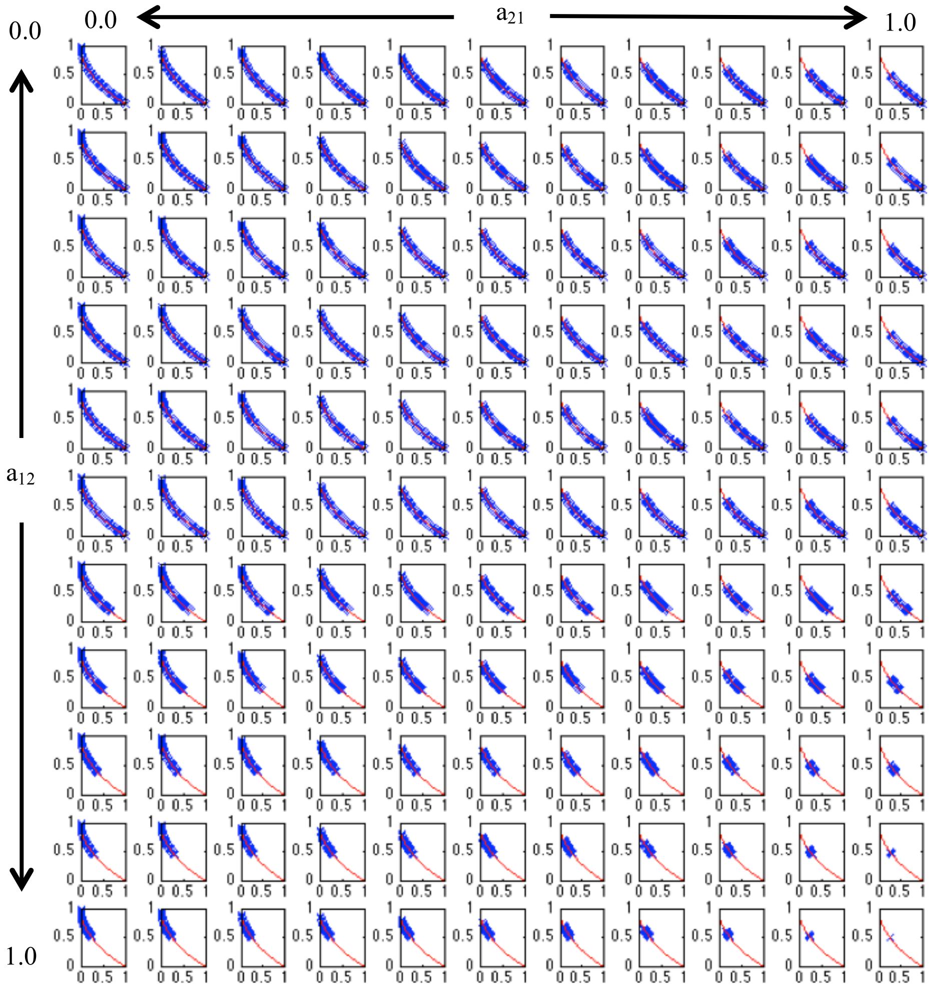
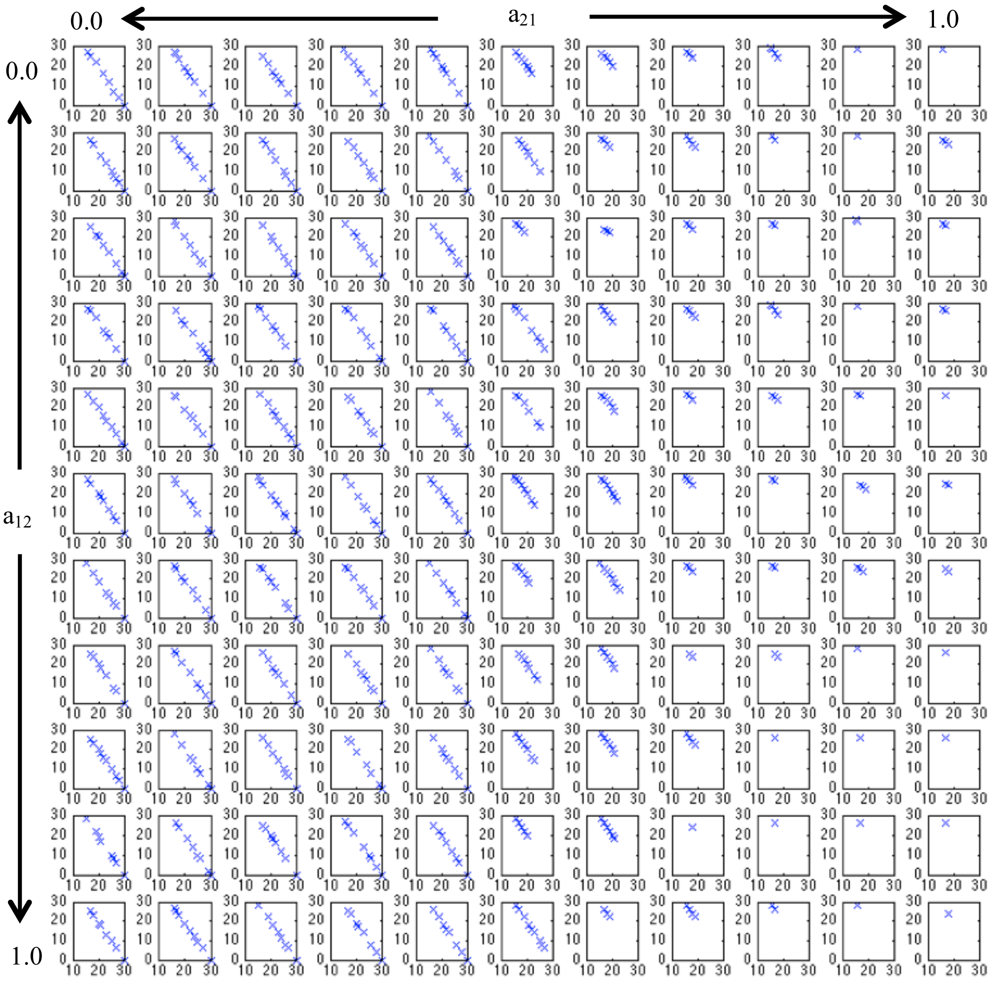


Figure 6.9. The effect of applying different weights to objectives on G-MOEA, when optimising the 30-bit *unitation versus pairs* problem. For each subplot, f_1(x) is on the x-axis and f_2(x) is on the y-axis. Blue crosses show solutions. Weight a_{21} increases from 0.0 to 1.0 in increments of 0.1, from subplots from the left to the right. Weight a_{12} increases from 0.0 to 1.0 in increments of 0.1, from subplots from the top to the bottom.



6.6.3.3 R-NSGA-II

As with NSGA-II and G-MOEA, R-NSGA-II is tested on the ZDT1 problem. The same configuration is used as with other two algorithms, a population size of 100 evolved for 500 generations. Three reference points are tested, without any additional weightings: (0.1,0.9), (0.5,0.5) and (0.9,0.1) and ε was set to 0.01. The reference points are purposefully put slightly away from the Pareto front in order to see if Pareto optimal solutions can still be found.

Figure 6.10 shows the results of using each of these reference points after 500 generations. Even though the reference points are in suboptimal regions in space, the search still finds solutions on or very close to the Pareto front. For each of the tested reference points, search navigates to a region very close to the reference point. On this task, R-NSGA-II appears to successfully be able to reflect user preferences.

The algorithm was also tested on the 30-bit *unitation versus pairs* problem, with the population size again set to 10 and evolved for 500 generations. Figure 6.11 shows that R-NSGA-II is successful on this discrete problem. Three reference points were tested, without any additional weighting: (15,30), (24,15) and (30,0). In each case, the algorithm was able to find the Pareto optimal solution closest to the reference point. It also found other Pareto optimal solutions, skewed towards the reference point in question.

6.6.4 Discussion

The examples presented above give us an insight into NSGA-II and the G-MOEA and R-NSGA-II adaptations. Through the two example problems, NSGA-II is shown to perform well on both a continuous and a discrete problem. On the discrete problem the algorithm was able to capture the entire Pareto front, while on the continuous task it made a close approximation of the true front. This suggests that it is a good candidate for Tool Selection and Machining Parameter optimisation. It has been employed on this latter problem by (Datta and Deb, 2009; Yang and Natarajan, 2010).

Both R-NSGA-II and G-MOEA were shown to be able to guide search towards different regions of the search space. G-MOEA appears to work better on the continuous problem than the discrete one, which could be due to falling into local optima or difficulty in finding the correct weights. The same weight configurations did not have the same affect on the search regions on the different problems. R-NSGA-II was able to guide search similarly on both the continuous and the discrete problem. This suggests that it might be better suited to discrete Tool Selection problems. One issue with this method is that a specific reference point needs to be set before search begins, which might be difficult to determine a priori. G-MOEA has the advantage that it can be set to relative differences among objectives.

Figure 6.10. Graphs showing the results of using R-NSGA-II on ZDT1 with reference points (a) [0.1,0.9], (b) [0.5,0.5], and (c) [0.9,0.1]. The reference points are shown with a green circle, the Pareto optimal front is shown in red and solutions found after 500 generations are shown with blue crosses.

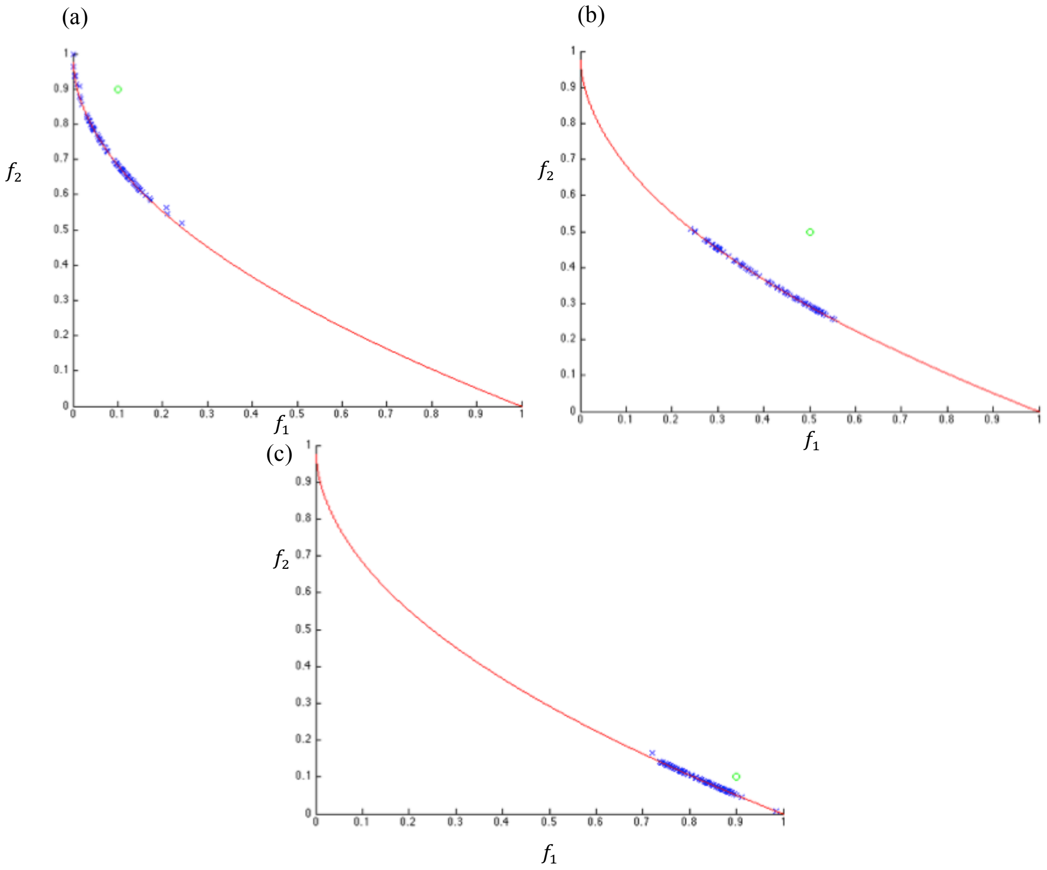
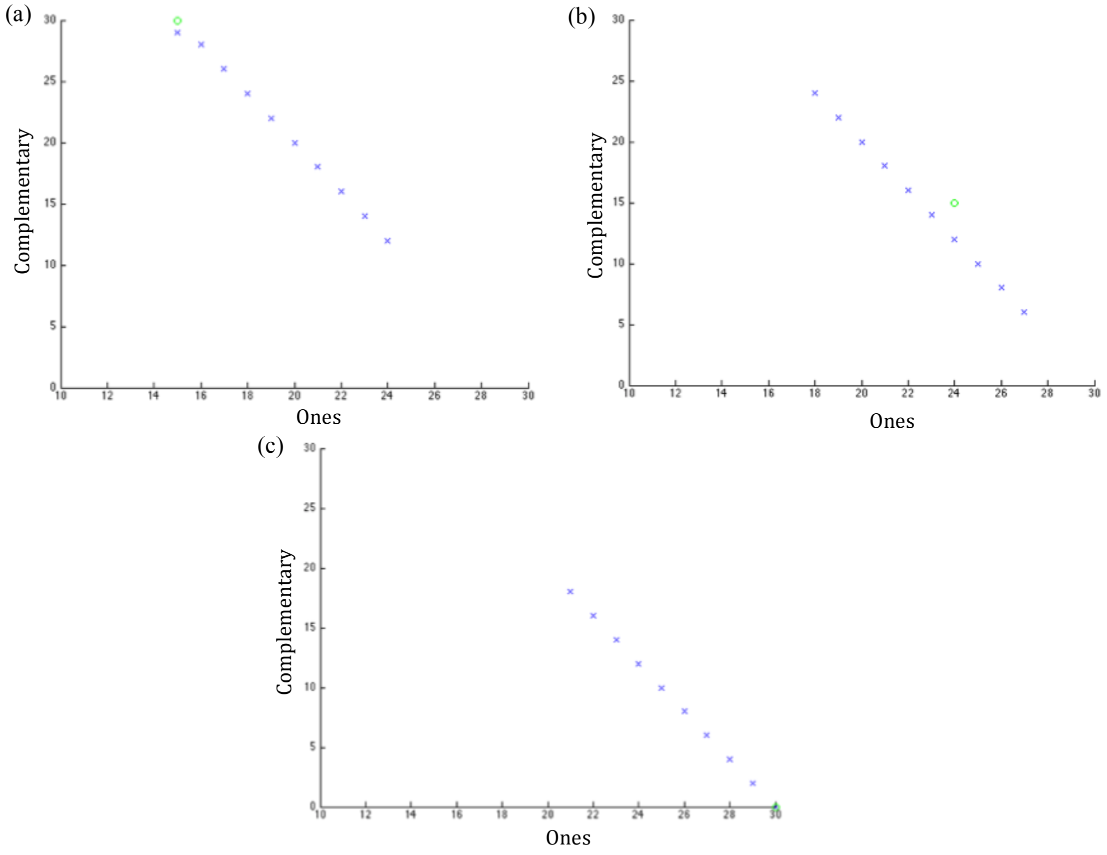


Figure 6.11. Graphs showing the effect of applying R-NSGA-II to the 30 bit unitation versus pairs problem with three different reference points: (a) [15,30] (b) [24,15] and (c) [0,30]. Solutions found at the end of 500 generations are shown with blue crosses. The reference point is shown with a green circle.



6.7 Multi-objectivisation and Guided Elitism

In this chapter we have introduced multi-objective search and explored several multi-objective algorithms. In this section we will look at the application of multi-objective techniques to single-objective problems, which can provide more effective search. This could have applications for solving difficult machining problems. We first discuss the main concepts behind multi-objectivisation, before introducing a novel adaptation to NSGA-II, *Guided Elitism*, which provides preferential search in a *multi-objectivised* search environment. This novel algorithm is then compared to NSGA-II and a single-objective Genetic Algorithm on the 0/1 Knapsack Problem.

6.7.1 Background

Multi-objectivisation was first introduced in (Knowles et al., 2001) as a method for reducing local optima in single-objective problems through the implementation of multi-objective techniques. This first attempt showed that multi-objectivised algorithms could provide more efficient solutions to the Hierarchical IF and Only IF Problem (HIFF) and the Travelling Salesman Problem (TSP). The process of multi-objectivisation is described using a simple Hill Climbing algorithm. A Hill Climber will attempt to find a state one move away that improves on its current state. However, to reach the global optimum, it may be necessary to move through solutions that are worse in a local sense but provide a route to higher value states. Using Pareto dominance rather than a single value to evaluate solutions, the Hill Climber is able to move across solutions that are 'incomparable' in a Pareto sense. This can allow the Hill Climber more freedom to explore the search space.

In (Lochtefeld and Ciarallo, 2012), multi-objectivisation methods are analysed and shown to reduce the signal-to-noise ratio and break epistasis. By evaluating solutions using multiple objectives, their good aspects can be isolated from the 'noise' of their undesirable characteristics. This noise can be exacerbated by the use of penalty functions and constraint handling techniques. These benefits are echoed in (Jensen, 2004), who describes how multi-objectivisation improves the identification of good building blocks and also assists in maintaining diversity in the population.

Multi-objectivisation is typically achieved through two main ways: decomposition and helper objectives. With both methods, it is paramount that the optimal solution under the single objective method occurs on the newly created Pareto front. Decomposition breaks up a function into sub-functions (Knowles et al., 2001; Garza-Fabre et al., 2012). An example of this can be seen in the TSP. To multi-objectivise the original objective function of summing the distances between all the cities in a tour, this could be split into the sums of the distance travelled in two sub tours (Knowles et al., 2001). Many existing single-objective problems have fitness functions that are formed as an aggregated weighted sum of the objectives. These can easily be decomposed by considering each objective individually, in a Pareto fashion.

Many problems have constraints that could be considered as objectives, so multi-objectivisation could be achieved through constraint relaxation (Coello Coello et al., 2007). For example, an optimisation task could be to,

maximise $f(x)$, subject to c < 10

This could be multi-objectivised by having two objectives,

Objective 1: maximise $f(x)$

Objective 2: maximise $-c$

Both the constraint relaxation and aggregated fitness decomposition methods can result in poor search because infeasible or undesirable solutions can become Pareto optimal and also the interaction between objectives can be lost (Coello Coello et al., 2007; Lochtefeld and Ciarallo, 2012). To improve search performance, it could be beneficial to bias search towards the feasible region (Runarsson and Yao, 2005).

Helper functions can be used in addition to the original single objective function. These can include a number of aggregated objectives or decompositions of the original function, in addition to the original objective function (Jensen, 2004; Lochtefeld and Ciarallo, 2012). Another simple helper function method is to create a second objective by adding noise to the original objective function (Watanabe and Sakakibara, 2005). (Jensen, 2004) found that the use of helper functions can significantly improve on the performance of a single-objective GA but that the number of extra objectives needed to be kept low. In (Lehman et al., 2013) novelty is used as an additional objective and is shown to improve robot navigation in deceptive maze problems.

Researchers have shown that multi-objectivisation has outperformed single-objective search on an array of problems (e.g. Knowles et al., 2001; Deb and Saha, 2010; Garza-Fabre et al., 2012). However, (Ishibuchi and Nojima, 2007) have found the reverse situation occur when the optimal solution lies on the edges of the Pareto front and also when there are many objectives that need to be decomposed, which leads to the many-objective problems discussed earlier in Section 6.3.4.

6.7.2 Guided Elitism

Guided Elitism is a novel adaptation of NSGA-II that acts as a single/multi-objective hybrid. The motivation behind it is to create a better balance between exploration and exploitation in the context of constraint relaxation based multi-objectivisation. By using Pareto-based multi-objective search, the goal is to find a diverse spread of Pareto optimal solutions. This benefits search by putting a larger emphasis on exploration, as we are searching for a set of solutions spanning multiple directions. Methods for encouraging a spread of solutions along the Pareto front can also help maintain diversity in the population, which can help prevent search from getting stuck in local optima. However, as described above, this also carries with it a large problem - the number of ‘optimal’ solutions has increased. This means that search is concentrating, at least partly, on infeasible areas and undesirable solutions, which could prevent the optimal single-objective solution from being found, given a limited amount of resources. Coello Coello et al. suggest that when using multi-objectivisation through constraint relaxation there may be a need to ‘guide search’ (2007), which is the goal of the Guided Elitism algorithm.

Guided Elitism (GE) adds preferences to search using a single-objective function. The aim of GE is to combine the elements of the free, unconstrained NSGA-II with a small amount of extra selective pressure in the region desired, that can hopefully allow desired results to be returned, while preventing search from getting stuck in local optima. In the context of the exploration/exploitation paradigm, GE combines the exploration power of the multi-objective algorithms with the exploitation power of weighted aggregate functions.

The adaptation works in the following way. At the point in NSGA-II where an enlarged population is created by merging the previous generation with a population of new children, in GE the entire population is sorted according to the single-objective evaluation function. The best *y* unique members are removed from this temporary population and added into the population of the next generation. Their dominance ranking is set to 1 (the highest value) and their crowding distance is set to their single-objective fitness value, making sure it is higher than any other crowding distance score (e.g. by adding 1 if the crowding distance is normalised between 0 and 1). This means that these elite solutions are given guaranteed survival and will also win in a binary tournament against ‘non-elite solutions’. If two ‘elite’ solutions are compared in a binary tournament, the one with the highest single-objective fitness will win. Once the limit of *y* individuals has been reached, the other solutions are added to the new population using the normal NSGA-II methods (Pareto ranking and Crowding Distance). This means that extra selective pressure is applied in two places, deciding which members to keep in the next generation and in selecting parents for generating new children. Pseudo-code for the adaptation can be found in Algorithm 6.5, replacing lines 8 -15 in Algorithm 6.3 (NSGA-II main code).

Although GE has the potential to create more selective pressure in a desired region while maintaining a large exploration of objective space, there are two additional parameters that must be set compared to the original NSGA-II. The first is the single-objective function. In the case of multi-objectivisation this is simple because one can apply the same function used to evaluate the single-objective version of the problem. However, in a multi-objective case, the choice of objective function needs to be carefully chosen to bias search towards the correct region. The second parameter is *y*, the number of solutions to make ‘elite’. This affects the balance between exploration and exploitation. In this thesis we use a value of 10% of the population size, as we wanted to ensure that the best solution from the single-objective perspective would be protected but encourage the discovery of a wide range of solutions.

6.7.3 Multi-Objectivisation of the 0/1 Knapsack Problem with Guided Elitism

In this section we compare a single-objective Genetic Algorithm (GA) with NSGA-II and NSGA-II with GE on a benchmark task, the 0/1 Knapsack Problem. The problem, which shares similarities to Tool Sequence Optimisation, is multi-objectivised through constraint relaxation. The algorithms are tested to see if there is benefit to multi-objectivisation and if the novel Guided Elitism adaptation provides better single-objective solutions than NSGA-II.

Algorithm 6.5: Pseudo-Code for Guided Elitism applied to NSGA-II

```
p_r = p + p'
initialise new_population as an empty list
sort p_r using the single objective evaluation function
while size(new_population) < y, do
    remove top ranked members from p_r and add to new_population
Pareto rank the modified population, p_r
assign crowding distances to the modified population, p_r
re-sort p_r according to Pareto rank and crowding distance
while new_population < pop_size, do
    remove top ranked members of p_r and add to new_population
end while
```

6.7.3.1 The 0/1 Knapsack Problem

The Knapsack Problem is a famous combinatorial optimisation problem. We have access to a set of items, each having an associated value and a weight. Given a knapsack that can only withstand a certain weight, the dilemma is to find a transportable subset of items that maximises the total profit. There are many varieties of this problem, including using multiple knapsacks and repeatable items. In the 0/1 version, each item can only be used once (Michalewicz and Arabas, 1994). Here, given a set of weights, $W = [w_0, w_1, \dots, w_n]$; values, $V = [v_0, v_1, \dots, v_n]$; and a capacity, C, the task is to find the binary vector, $\boldsymbol{x} = [\boldsymbol{x}_0, \boldsymbol{x}_i, \dots, \boldsymbol{x}_n], \boldsymbol{x} \epsilon [0,1]$, where

$$f_v(\boldsymbol{x}) = \sum_{i=0}^{n} v_i \,.\, x_i \qquad (6.13)$$

is maximum and $f_w(x) \le C$ where

$$f_w(x) = \sum_{i=0}^{n} w_i \,.\, x_i \qquad (6.14)$$

The 0/1 Knapsack problem and other variants have been explored by a large number of researchers in the Evolutionary Computation community. The problem has been used to test constraint-handling techniques, for example in (Michalewicz and Arabas, 1994); multi-objective optimization by extending the problem to multiple knapsacks, for example in (Grosan, 2004); and multi-objectivisation (Watanabe and Sakakibara, 2005). Similarly to the last authors, the work presented in this section also tackles multi-objectivisation of the 0/1 Knapsack problem, taking a different angle by working in a completely unconstrained environment with the inclusion of preferential search.

6.7.3.2 Multi-objectivisation through Constraint Relaxation

The capacity of the knapsack in 0/1K, similarly to the remaining stock in Tool Sequence Optimisation (TSO), is treated as a constraint but it could also be considered as an extra objective. In 0/1K, the problem could be reformulated as a multi-objective problem containing two objectives, (1) maximise total value and (2) minimise total weight. The optimal solution in the single-objective version will also be Pareto optimal because it will have the highest total value for its specific weight class.

There are two potential benefits in applying multi-objectivisation to 0/1K, which also apply to TSO. The first is that search may be able to avoid getting stuck in areas of local optima. The second is that a selection of solutions can found. While the former is an obvious reason to use multi-objective techniques, the latter could also prove to be useful in real-world applications. Imagine 0/1K in a real world situation. Given a capacity of 50kg, we may discover one solution with properties (£1000, 50kg) and another with (£995, 30kg). If we have to take a 25-mile hike across the desert to return with our bounty we may decide that the second solution is a better option! Finding several solutions in a single search also gives the decision maker the option to make a choice based on extra information that may not be included in the optimisation task in order to reduce the complexity of search. If we are collecting shells in our knapsack, we may choose between two differing sets of similar value because we find certain items more aesthetically pleasing. In TSO there could be many benefits in having a selection of solutions. For example, the decision maker may decide that a tool sequence with marginally higher total time and a much lower surface tolerance is preferable because of a simpler finishing process.

6.7.3.3 Summary of Experiments

A number of 0/1 Knapsack Problem experiments are presented below, using randomly generated item sets of various sizes. Each knapsack problem is used to evaluate the performance of a GA, NSGA-II and NSGA-II with the GE adaptation on the single objective 0/1K, as described by eq. (6.13) and (6.14). The goal of the experiments is to see if multi-objectivisation improves search performance, and whether guiding search helps negate problems created by constraint relaxation.

The size, S, of the knapsacks range from 20 – 200. Each knapsack is tested with three capacities, equal to $10S$, $20S$, and $30S$. The weights and values of the items were randomly generated, from a uniform distribution of integers between 1 and 150, with no correlation between the two.

6.7.3.4 Algorithmic Parameters

The algorithms all used the same binary representation and genetic operators. Each item is assigned to an element of the binary string, and if the element is 1 the item is included in the knapsack. There is a 0.3 probability of a single point crossover, with the length and point determined stochastically. For the mutation operator, each bit of a generated individual has a $\frac{1}{S}$ probability of being flipped. The GA and GE algorithms both use the same single objective function, $f_s(\boldsymbol{x})$, which is the value of the knapsack with a punishment, $f_p(\boldsymbol{x})$, for being over the weight limit:

$$f_s(\boldsymbol{x}) = -(f_v(\boldsymbol{x}) + f_p(\boldsymbol{x})) \qquad (6.15)$$

$$f_p(\boldsymbol{x}) = \frac{f_w(\boldsymbol{x}) - C}{10}, if \; f_w(\boldsymbol{x}) > c, else\, f_p(\boldsymbol{x}) = 0 \quad (6.16)$$

Guided Elitism is applied to 10% of the population. NSGA-II and GE both use the same multi-objective evaluation functions with no punishment factors. There are two objective functions:

$$f_1(\boldsymbol{x}) = -f_v(\boldsymbol{x}) \qquad (6.17)$$

$$f_2(\boldsymbol{x}) = f_w(\boldsymbol{x}) \qquad (6.18)$$

The value of a knapsack in 6.15 and 6.17 is multiplied by -1 in order to frame the problem as a minimisation task.

6.7.3.5 Results

To evaluate the performance of multi-objectivisation on the Knapsack Problem, we test a single-objective GA, NSGA-II and NSGA-II with Guided Elitism (GE) on 19 different knapsacks of increasing size, with experiments on 3 different capacities per knapsack. Each experiment was repeated for 100 trials, and the algorithms run for 200 generations with a population size of 100.

The algorithms are assessed solely on the best solution found from a single-objective perspective, the solution with the highest value and a weight under the capacity. This allows a fair comparison between the single and multi-objective algorithms. Tables 6.1 and 6.2 show the median and interquartile ranges for the best solutions found by the three algorithms, on the 19 knapsacks. The boxplots in Figure 6.13 and Figure 6.14 show the distribution of solutions found for each experiment, across the 100 trials from knapsacks with 50 – 200 items. We can see that all of the algorithms perform well on the task. They are all able to find good, feasible solutions, close to the optimum, and the variation between individual trials is small.

The results clearly show that as the search space becomes larger both NSGA-II and GE outperform the GA. The GA performs comparably until knapsack problems with 40 items and over. This is interesting because it suggests that using a multi-objective approach can produce better solutions on this single-objective problem. NSGA-II is able to find good, feasible solutions, without using any constraint handling techniques at all. This means that it is able to freely explore the search space with none of the negative affects of using a punishment factor. There are three examples where NSGA-II finds a better overall solution than GE. These can be seen in Figure 6.13 with a problem size of 100 items and capacity of 100, and in Figure 6.14 with 170 items and capacity of 1700, and 200 items and capacity of 2000. This could suggest that the search bias in GE may affect its ability to find optimal solutions.

Apart from these isolated best scores examples, GE has by far the most successful search performance for every knapsack size and capacity limit. It has an equal or higher median best score than the other algorithms in every case, and relatively low variance between results. Compared to the GA, it not only finds better solutions on average but looking at the boxplots in Figure 6.13 and Figure 6.14, we can see that from knapsacks with 80 items or more, it also finds solutions that the GA cannot. This suggests that this multi-objective hybrid is able to avoid local optima better than the GA.

Figure 6.12. Showing in (a) and (b) differences in Empirical Attainment Function for an example NSGA-II and Guided Elitism (GE) on a 0/1 Knapsack Problem with 100 items and a capacity of 1000, over 100 runs. Showing (a) the full range of solutions obtained and (b) solutions obtained in the region of negative value between (-3,600,-3,200) and weight between (900,950). The legend shows the frequency that one algorithm found solutions in that region that dominated the other. The solid lines show the best and worst attained surfaces, while the dashed line shows the median attained surface. Showing in (c), solutions returned from an example run of NSGA-II and GE, in the region of negative value between (-3,600,-3,200) and weight between (900,950).

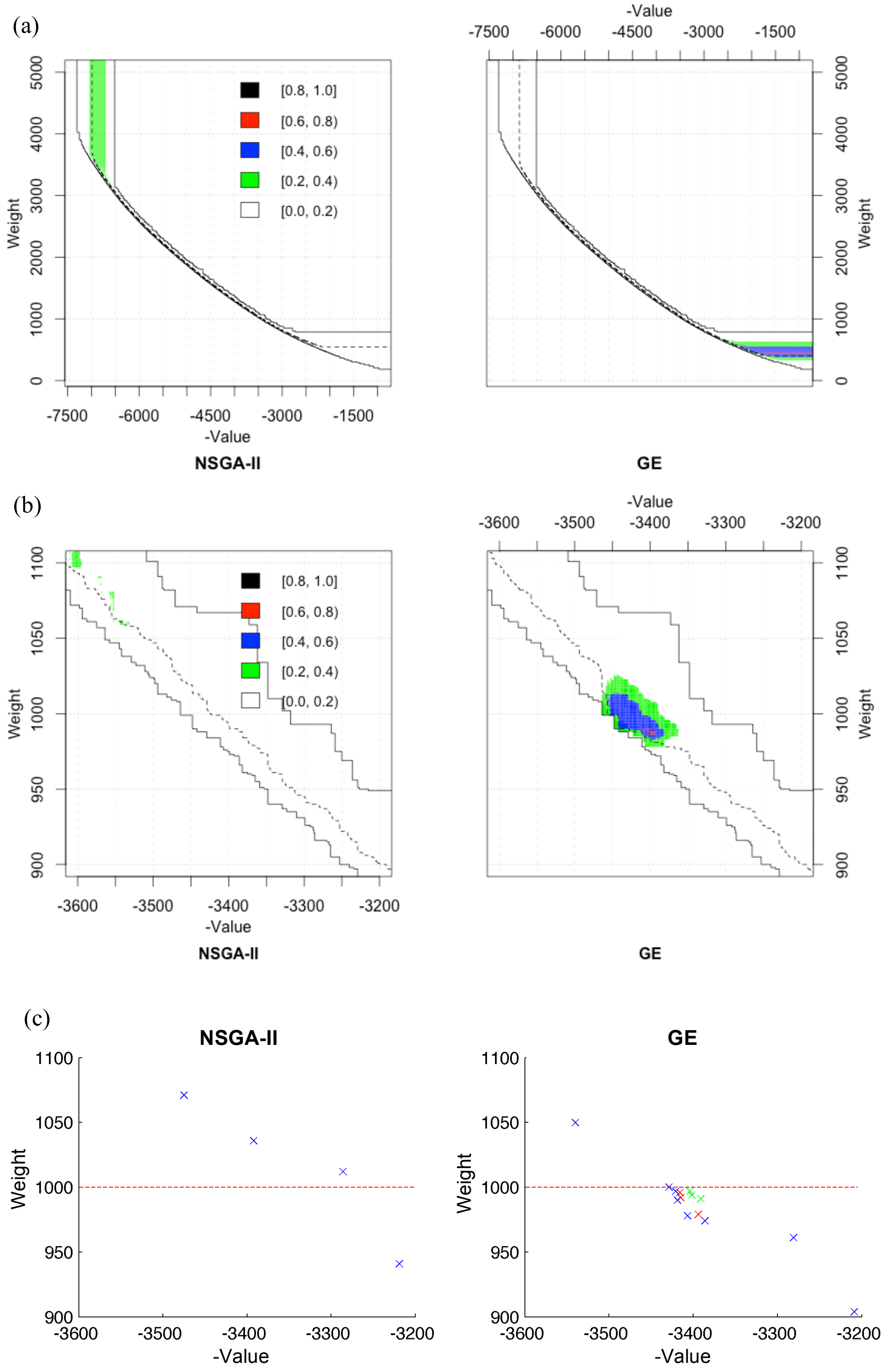


Table 6.1. The median and interquartile ranges (IQR) for the best solution found <= to the capacity for a 0/1 Knapsack problem of sizes 20 – 100. Showing results for the NSGA-II algorithm, Guided Elitism (GE) and a Genetic Algorithm (GA). The capacity C_1 indicates 10 times; C_2, 20 times; and C_3, 30 times the knapsack size. A Kruskal-Wallis test for equal distributions was applied. If it indicated that there were significant differences between at least two algorithms a pairwise Wilcoxan rank sum test was performed. Superscripts indicate that there were not significant differences (p < 0.01) between the named algorithms following the Bonferroni correction for multiple comparisons.

Size	Capacity	Optimum	Median			IQR		
			NSGA-II	GE	GA	NSGA-II	GE	GA
20	C_1	450	**450.0** $^{\text{GE,GA}}$	**450.0** $^{\text{GA,NSGA-II}}$	**450.0** $^{\text{NSGA-II,GE}}$	0.0	0.0	0.0
	C_2	710	**710.0** $^{\text{GE,GA}}$	**710.0** $^{\text{GA,NSGA-II}}$	**710.0** $^{\text{NSGA-II,GE}}$	0.0	0.0	0.0
	C_3	910	**910.0** $^{\text{GE,GA}}$	**910.0** $^{\text{GA,NSGA-II}}$	**910.0** $^{\text{NSGA-II,GE}}$	0.0	0.0	0.0
30	C_1	914	905.0	**914.0** $^{\text{GA}}$	**914.0** $^{\text{GE}}$	14.0	0.0	9.0
	C_2	1374	1365.5 $^{\text{GA}}$	**1374.0**	1369.0 $^{\text{NSGA-II}}$	19.0	0.0	20.3
	C_3	1660	**1660.0** $^{\text{GE,GA}}$	**1660.0** $^{\text{GA,NSGA-II}}$	**1660.0** $^{\text{NSGA-II,GE}}$	0.0	0.0	0.0
40	C_1	1354	1341.0 $^{\text{GA}}$	**1354.0**	1338.0 $^{\text{NSGA-II}}$	32.0	3.0	27.0
	C_2	2068	2057.0 $^{\text{GA}}$	**2068.0**	2050.0 $^{\text{NSGA-II}}$	15.0	0.0	23.5
	C_3	2614	2599.0	**2614.0**	2592.0	15.0	0.0	23.0
50	C_1	1991	1955.0 $^{\text{GA}}$	**1983.0**	1944.5 $^{\text{NSGA-II}}$	35.5	12.0	37.3
	C_2	3047	3014.0	**3047.0**	2997.0	30.5	9.0	29.5
	C_3	3817	3784.5	**3817.0**	3769.0	40.3	13.0	30.0
60	C_1	1669	1618.0	**1659.0**	1590.5	37.5	15.0	45.5
	C_2	2727	2698.0	**2727.0**	2649.5	35.3	11.0	43.3
	C_3	3458	3427.0	**3451.0**	3413.0	29.3	8.5	26.5
70	C_1	2054	2002.5	**2043.5**	1957.0	40.3	13.3	35.0
	C_2	3144	3088.5	**3129.0**	3064.0	45.5	18.0	41.8
	C_3	4018	3967.5	**3999.0**	3948.0	37.8	11.0	37.0
80	C_1	2357	2274.0	**2331.0**	2216.0	55.5	29.8	63.5
	C_2	3675	3600.0	**3655.0**	3562.5	48.5	21.0	41.0
	C_3	4686	4606.0	**4657.0**	4566.5	44.5	22.5	51.5
90	C_1	2782	2691.5	**2753.5**	2609.5	59.8	27.3	70.3
	C_2	4180	4095.0	**4138.0**	4013.5	49.3	20.5	53.8
	C_3	5218	5141.5	**5195.5**	5101.5	42.8	21.3	43.8
100	C_1	3464	3320.5	**3383.0**	3231.0	69.0	34.3	70.5
	C_2	5195	5091.0	**5145.0**	5021.5	66.0	26.0	63.5
	C_3	6486	6386.5	**6441.5**	6334.0	46.0	29.3	50.0

Table 6.2. The median and interquartile ranges (IQR) for the best solution found <= to the capacity for a 0/1 Knapsack problem of sizes 110 – 200. Showing results for the NSGA-II algorithm, Guided Elitism (GE) and a Genetic Algorithm (GA). The capacity C_1 indicates 10 times; C_2, 20 times; and C_3, 30 times the knapsack size. A Kruskal-Wallis test for equal distributions was applied. It indicated that there were significant differences between at least two algorithms on all experiments. A pairwise Wilcoxan rank sum test found that there were significant differences ($p < 0.01$) between all algorithms following the Bonferroni correction for multiple comparisons.

Size	Capacity	Optimum	Median			IQR		
			NSGA-II	GE	GA	NSGA-II	GE	GA
110	C_1	4074	3913.0	**3975.5**	3789.0	99.5	51.3	109.0
	C_2	6086	5948.5	**6027.5**	5819.5	81.5	39.5	82.0
	C_3	7426	7317.5	**7370.5**	7234.5	50.8	31.3	56.0
120	C_1	4069	3875.0	**3961.0**	3780.0	88.3	55.8	114.0
	C_2	5979	5821.0	**5897.0**	5746.5	63.8	36.5	73.5
	C_3	7481	7332.5	**7399.5**	7259.5	64.5	33.8	69.0
130	C_1	5465	5220.0	**5326.0**	5042.5	93.5	65.5	143.5
	C_2	7535	7369.0	**7451.0**	7261.5	67.3	49.5	88.8
	C_3	8829	8687.0	**8747.0**	8598.0	59.0	38.8	87.0
140	C_1	5135	4827.5	**4956.0**	4671.0	142.3	80.3	127.5
	C_2	7653	7408.0	**7500.5**	7288.0	88.5	52.8	105.8
	C_3	9547	9330.5	**9413.0**	9242.5	98.3	51.8	78.5
150	C_1	5184	4890.5	**4993.0**	4738.5	112.8	64.5	176.5
	C_2	7642	7388.0	**7487.0**	7248.5	97.5	70.0	93.3
	C_3	9379	9162.5	**9244.0**	9085.5	88.0	58.5	95.5
160	C_1	5798	5380.5	**5502.0**	5161.0	128.0	104.3	109.5
	C_2	8614	8326.0	**8404.5**	8113.5	123.5	78.3	126.8
	C_3	10693	10391.5	**10508.0**	10275.0	101.8	66.3	110.0
170	C_1	6168	5734.5	**5831.0**	5507.5	144.5	100.8	169.8
	C_2	9038	8695.0	**8786.5**	8492.5	119.0	78.0	129.3
	C_3	11051	10727.5	**10866.0**	10644.0	108.8	60.3	99.5
180	C_1	6142	5683.0	**5806.5**	5529.5	163.8	95.0	161.0
	C_2	9167	8795.5	**8915.0**	8619.0	136.3	98.8	116.8
	C_3	11271	10907.5	**11022.0**	10813.5	111.8	84.8	90.3
190	C_1	6914	6334.5	**6483.0**	6185.0	169.3	94.3	153.0
	C_2	10279	9846.0	**9955.0**	9648.5	132.8	91.5	122.0
	C_3	12818	12356.0	**12495.5**	12246.5	138.8	82.5	105.0
200	C_1	6902	6215.0	**6375.5**	6033.0	175.3	143.5	182.0
	C_2	10332	9864.0	**9992.0**	9664.0	165.0	105.5	114.5
	C_3	12805	12358.0	**12482.5**	12263.5	125.8	92.8	127.5

Figure 6.13. Boxplots showing the results of the best solution found from 100 trials of NSGA-II (N), Guided Elitism (GE) and a Genetic Algorithm (GA) for knapsack problems with three capacities and sizes from 50 – 120 items (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range).

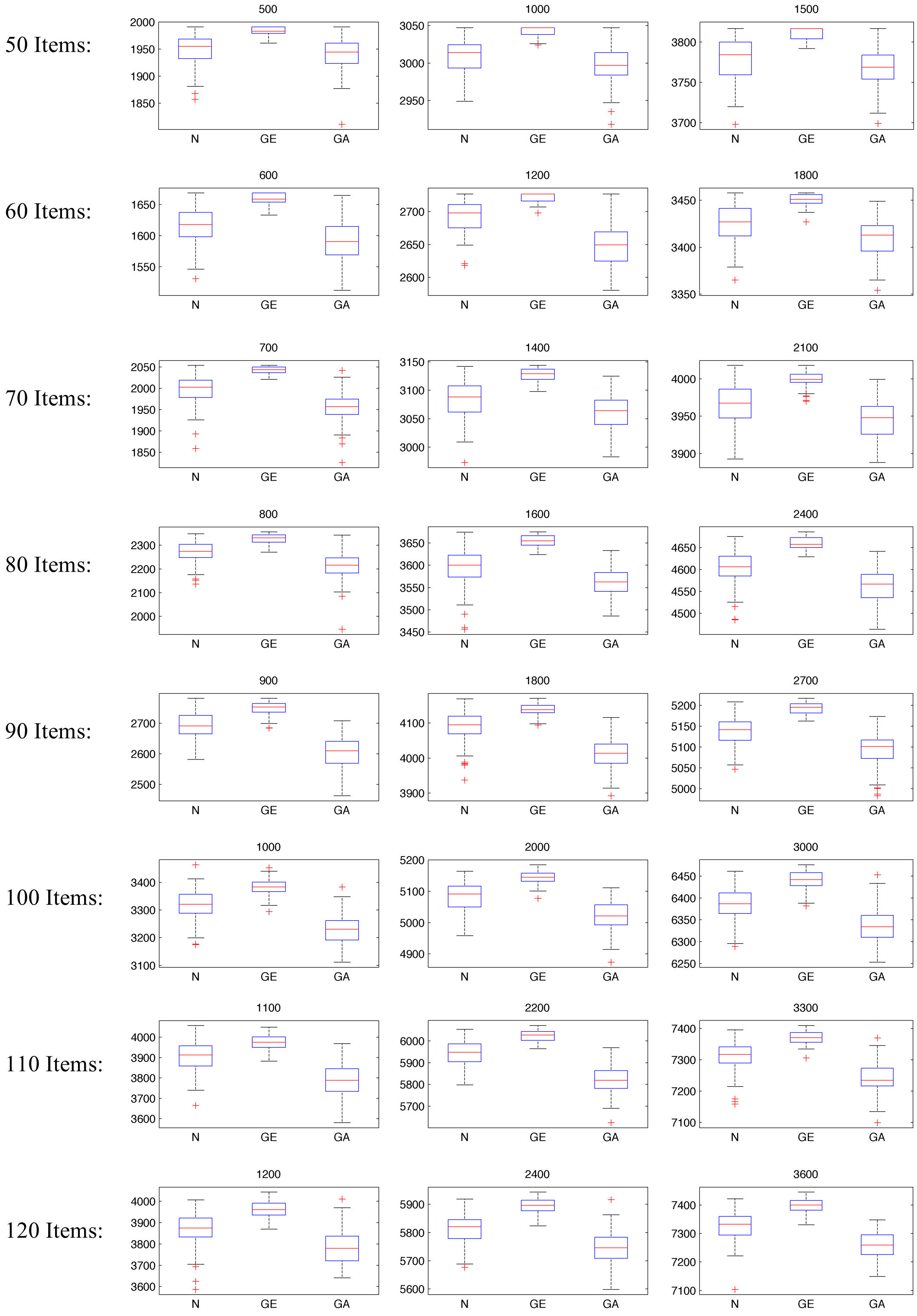
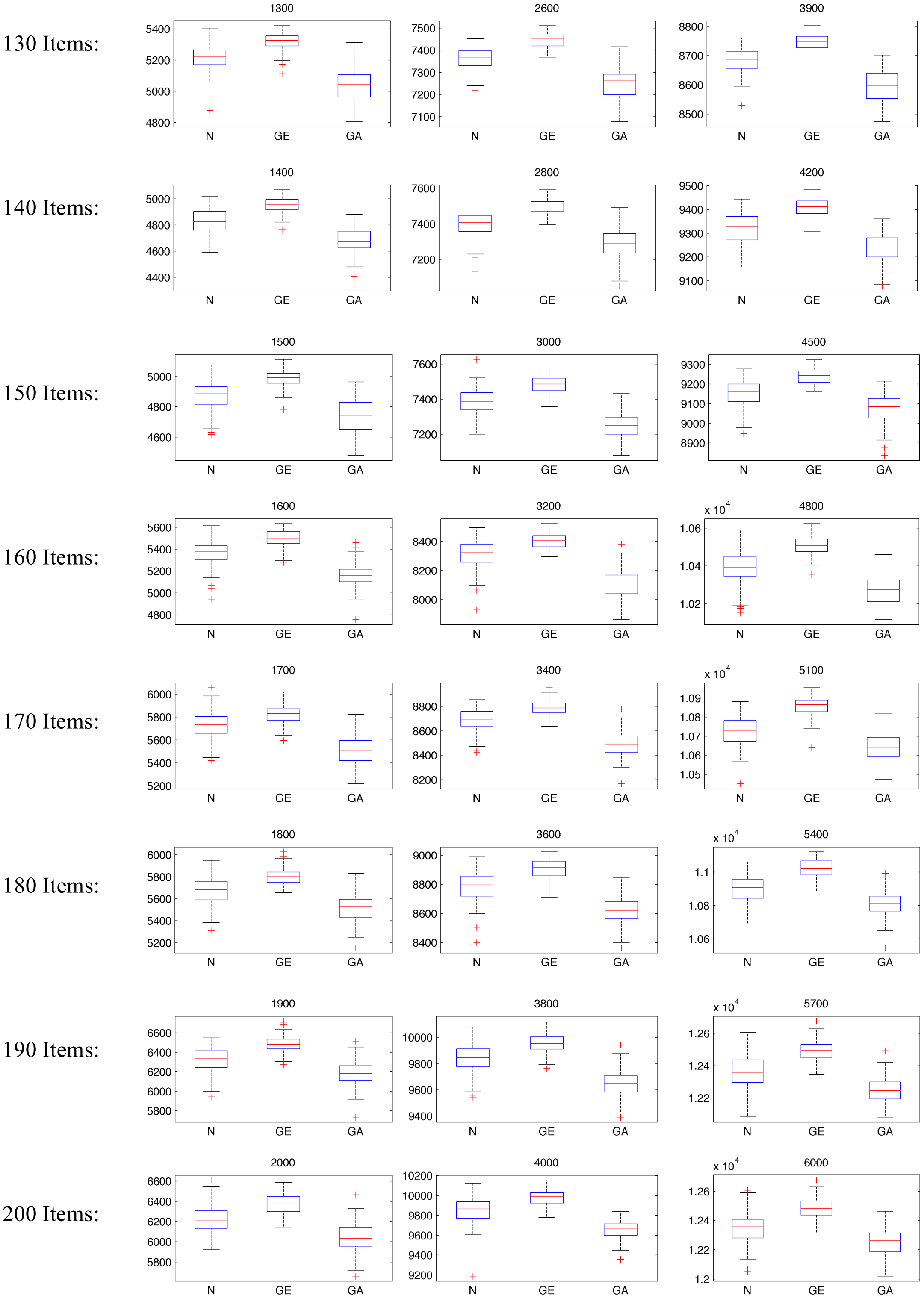


Figure 6.14. Boxplots showing the results of the best solution found from 100 trials of NSGA-II (N), Guided Elitism (GE) and a Genetic Algorithm (GA) for knapsack problems with three capacities and sizes from 130 – 200 items (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range).



6.7.3.6 Comparing the Pareto Fronts of NSGA-II and Guided Elitism

As well as obtaining good solutions from a single-objective perspective, the multi-objective NSGA-II and GE algorithms maintain a population of diverse solutions. To gain an insight into the performance of the two algorithms, Figure 6.12 shows the differences in Empirical Attainment Functions (as described earlier in Section 6.2.2.2) between NSGA-II and GE, for a knapsack with a size of 100 items and a capacity of 1000. From this we can see that both algorithms find a diverse range of solutions, although on average they are much better at finding solutions in the middle, rather than the side regions of the discovered Pareto front. This shows a great deal of diversity in the population. From Figure 6.12(a), we can see that NSGA-II appears to more frequently discover better solutions to the top left of the front, while GE finds better solutions towards the bottom right.

In Figure 6.12(b), we zoom in on the region where the desired solution is, near the capacity limit of 1000. Here, we can clearly see the effect of the GE adaptation, as the algorithm finds dominating solutions here much more frequently. This is exemplified in Figure 6.12(c), where we show an example run from NSGA-II and GE. This shows that there are many more solutions clustered together in the desired region.

6.7.3.7 Discussion

We have introduced the Guided Elitism method to overcome problems with multi-objectivisation through constraint relaxation, tested on a benchmark problem. In the experiments above, the 0/1 Knapsack Problem was transformed into a multi-objective problem by considering weight and value to be separate, independent objectives. Both NSGA-II and GE outperformed the GA, which suggests that multi-objectivisation is a successful approach to take to finding solutions for this problem. This supports similar results found in (Watanabe and Sakakibara, 2005), who multi-objectivised the problem using a helper function. The first objective was the value plus a punishment factor based on the weight violation and the second was the value only. The difference between their work and the results presented above is that they controlled the constraint violation by using a repair function. In this work the investigation was centred on seeing how the algorithms worked in an unconstrained environment without using repairs, which can be extremely expensive computationally. Coello Coello et al. recommend using guidance when multi-objectivising single-objective problems through constraint relaxation (2007). Using the Guided Elitism method is found to improve search performance, while still maintaining good diversity in the population.

A method related to Guided Elitism can be found in (Ishibuchi et al., 2006). This work concentrates on probabilistically combining single-objective functions with Pareto-based algorithms to provide better selection pressure in the presence of many-objectives, which are difficult to solve with Pareto methods. A scalarising function is used with probability, *ps,* in the selection of individuals for mating, and with probability, *gs,* in the choosing individuals to keep from the enlarged population. The main differences between this algorithm and GE, is that in the latter case, survival is guaranteed, while in the former it is probabilistic. Using GE guarantees that we never lose our best solutions, from a single-objective perspective. These best solutions will also always win in a binary tournament, which adds selection pressure towards preferred regions of the search space. It would be difficult to achieve this effect using the probabilistic method.

The 0/1 Knapsack problem (0/1K) has similar features to Tool Sequence optimisation (TSO). The task in both is to optimise a single objective value, while satisfying a single constraint. In 0/1K we want to find a set of values that maximise profit while staying within a weight capacity. In TSO we want to find the sequence of tools that will minimise total machining time, while staying below a stock threshold.

Although 0/1K is a maximisation task and TSO is a minimisation task, it is easy to formulate 0/1K as a minimisation problem by multiplying the total value found by -1. Apart from this, the main difference is that in TSO there are precedence constraints and the ordering of tools affects the outcome of the objective function. Preceding tools in the sequence can also change the 'value' (time taken and/or stock remaining) of a tool. In 0/1K each item is independent and there is no ordering.

It is possible to multi-objectivise TSO in the same way that 0/1K has been treated in the experiments above, through relaxation of the minimum stock constraint. In Figure 6.12 (b) and (c) we can see that Guided Elitism finds several different dominant solutions close to the weight threshold. If similar results are found in TSO, this could prove useful to decision makers in real world situations.

6.8 Summary

In this chapter, the main concepts behind Multi-objective Optimisation have been introduced. The NSGA-II algorithm has been tested on a discrete and a continuous problem and shown to perform well on both. The reference point extension (R-NSGA-II) is also tested on these problems and found to provide a good method for preferential search in both problem domains. Multi-objectivisation was introduced, and is a process that can aid search by escaping local optima. An investigation was carried out into multi-objectivising the 0/1 Knapsack Problem through constraint relaxation. NSGA-II and a novel adaptation referred to as Guided Elitism are found to perform better than a Genetic Algorithm on knapsacks of various sizes. Guided Elitism performs better than NSGA-II, which supports suggestions by (Coello Coello et al., 2007) that guidance should be used when relaxing constraints.

A multi-objective approach can offer several benefits for optimisation in machining. In the previous chapter, we presented a tool sequence search space that the single-objective metaheuristic algorithms found particularly difficult to deal with because it contained several local optima. Multi-objectivising the problem could offer a way to escape areas of local optima, and it was shown to improve search performance on the 0/1 Knapsack problem. In addition, framing the Tool Sequence Optimisation in a Pareto fashion could enable the discovery of multiple solutions. This could prove useful to decision makers who may find themselves preferring different solutions depending on their current requirements. There are also extra objectives that are important to real world machining. One of these is tooling costs, and extending our system to incorporating this as a third objective could provide an even more useful selection of solutions. These multi-objective extensions are explored in the following two chapters.

Chapter 7

Multi-objectivisation of the Tool Selection Problem

In the previous chapter we saw that several researchers have found that using multi-objective techniques on single-objective problems can avoid convergence on local optima (Knowles et al., 2001; Watanabe and Sakakibara, 2005; Garza-Fabre, 2012). We also saw that multi-objective algorithms applied to the 0/1 Knapsack Problem performed better than a single-objective Genetic Algorithm (GA). In Chapter 5, we tested single-objective algorithms on a Tool Selection search space that was created through the hybridisation of two parts, which had many local optima and sharp jumps in fitness. In this situation we found that the tested single-objective algorithms struggled to consistently find the optimal solution. Increasing population size did improve search performance but this came at the expense of large evaluation numbers and frequent convergence on sub-optimal solutions.

In this chapter multi-objective algorithms are applied to the Tool Sequence Optimisation, testing on this difficult search space as well as on the most difficult other part seen in Chapter 5, to investigate whether multi-objectivisation provides improved search performance in the presence of several local optima. As we are dealing with computationally expensive evaluations, the algorithms are tested with strict evaluation budgets. Results from the experiments show that on the simpler search space the single-objective GA performs as well or better than the multi-objective algorithms, particularly with a low evaluation limit. However, on the search space with many local optima, the multi-objective algorithms are much more successful at consistently locating the optimal solution.

Incorporating preferential search through the Guided Elitism adaptation to NSGA-II introduced in the previous chapter and R-NSGA-II is shown to improve performance, particularly with the lower evaluation limits, although R-NSGA-II is shown to be sensitive to the choice of reference point. Algorithms using NSGA-II's Crowding Distance Assignment are shown to perform particularly badly with very small population sizes, and an adaptation is presented which addresses this. In addition to much more consistently returning the optimal solution in the presence of many local optima, the multi-objective algorithms also provide a set of solutions offering a trade-off between surface tolerance and machining time that could be useful to process planners.

This chapter is organised in the following way. First the motivation behind the multi-objectivisation approach is given. Next background is presented on how the Tool Selection Problem has been multi-objectivised and issues involving the preservation of desired solutions on the Pareto front are discussed. In Section 7.3, the methods used in the experiments are presented. Following this the results from the two experiments are presented in 7.4 and discussed in Section 7.5. Finally, we present a more detailed look at how the multi-objective algorithms perform in providing multiple solutions. Parts of this chapter appear in (Churchill et al., 2013a).

7.1 Introduction

The motivation behind the work presented in this chapter is to evaluate the effect of multi-objectivising the Tool Selection Problem. There are two main benefits that multi-objectivisation can offer here – improving the effectiveness of search and returning a selection of different solutions.

There are three main ways that a multi-objectivisation approach could help avoid local optima. Firstly, it could reduce the potentially disruptive effect of using a penalty function for constraint handling; secondly it could provide routes away from local optima by following multiple search directions; and finally, the techniques used in multi-objective algorithms such as NSGA-II to encourage a diverse spread of Pareto optimal solutions could help maintain diversity and prevent premature convergence.

The multi-objectivisation of the 0/1 Knapsack Problem (0/1K), presented in Section 6.8.3 in the previous chapter, showed that the multi-objective NSGA-II algorithm and the Guided Elitism adaptation (GE) outperformed a Genetic Algorithm on the single-objective problem. In this chapter we will apply the same techniques to the real world Tool Selection problem, and additionally test a second method for preferential search, R-NSGA-II. As with the experiments in Chapter 5, it is important to maintain a low computational cost on this problem and so the algorithms will be assessed under strict evaluations budgets.

Similarly to Chapter 6's work on the 0/1K, the multi-objective algorithms are shown to outperform single-objective metaheuristic techniques on the difficult search landscape. In doing so we provide a system that can work on search spaces with both small and large numbers of local optima, which should be encouraging for work in uncertain landscapes. Additionally we show that using preferential search can speed up convergence on the single-objective optimum, which could prove useful for other applications.

7.2 Background

In this section we discuss the multi-objectivisation of the Tool Selection Problem. First we describe how we can apply a similar process to multi-objectivising the 0/1 Knapsack Problem in the previous chapter by treating the main constraint as a second objective. Next we discuss important issues involving the preservation of the desired solution from a single-objective perspective when using small population sizes.

7.2.1 Multi-objectivising the Tool Selection Problem

As described in the previous chapter, multi-objectivisation has been shown to provide a more efficient exploration of a search space. The single-objective fitness function, f_{eval}, used by the algorithms in the previous chapters for a tool sequence, $\boldsymbol{E}$, and a part, p, is,

$$d = f_d(\boldsymbol{E}, p) \qquad (7.1)$$

$$f_{eval}(\boldsymbol{E}, p) = f_m(\boldsymbol{E}, p) + k$$

$$k = \begin{cases} 2k, & d > \mathrm{c} + \varepsilon \\ k, & \mathrm{c} < d \leq \mathrm{c} + \varepsilon \\ 0, & d < \mathrm{c} \end{cases} \qquad (7.2)$$

where f_m is a function that calculates the total machining time using tool path lengths (given in equation 3.2 from Chapter 3) and k is a punishment factor determined by the maximum thickness of excess stock, *d,* calculated using f_d.

In the previous chapter, the 0/1 Knapsack problem was presented with a similar single-objective evaluation function. It was successfully multi-objectivised by relaxing the weight constraint, removing the punishment factor and treating the constraint as a second objective. The same approach can be taken here, by relaxing the surface tolerance constraint and treating the maximum thickness of excess stock as a second objective. This gives the following two objective functions:

$$f_1(E, p) = f_m(\boldsymbol{E}, p) \qquad (7.3)$$

$$f_2(E, p) = f_d(\boldsymbol{E}, p) \qquad (7.4)$$

Our first objective function is the total machining time of a tool sequence and our second is the maximum distance of excess stock. Changing the optimisation problem in this way eliminates the need for the punishment term, *k*. Punishment factors are used to handle constraints but at the same time keep infeasible solutions within the population as they may have properties that help in finding good feasible solutions. However, their use can mean that a small change in genotype can have a large change in phenotype and neighbours in parameter space are distant in objective space. This can make it difficult for the metaheuristic algorithms to follow a search gradient (Bentley, 1999). Another question raised is how to choose a good value for k. It becomes another parameter that must be chosen in advance, and one that is difficult to set without prior knowledge of the search space. If it is too small, search will converge on undesirable solutions. If k is too large, search may be guided away from the optimal solution (Watanabe and Hashem, 2004). In the test search space created by the hybridisation of parts 2 and 4 presented in Chapter 5 (referred to below as Part 2-4-H), a solution could be incorrectly labelled as having excess stock outside of the surface tolerance constraint. This could also occur when using fitness approximation methods. Using a punishment factor could exacerbate these problems by driving search away from good regions of the search space.

In multi-objectivising the problem it is critical that the single-objective optimum solution occurs on the new Pareto front. In the formulation given by (7.3) and (7.4), the minimum of $f_{eval}(\boldsymbol{E}, p)$ (from 7.2) will be on the Pareto front, as the fastest solution found in each excess stock band will be Pareto optimal. Evaluating the problem using two objective functions offers extra information to the Evolutionary Algorithms, which could enable them to isolate good building blocks, which may be harder to find in an aggregate function

(Lochtefeld and Ciarallo, 2012). For example, solutions with low machining times may all contain a particular tool in the sequence. Even if some of these sequences have a large amount of excess stock, by separating the objectives the tool that contributes to a low machining time could be discovered and combined with a tool that achieves low excess stock, to create a very good feasible solution.

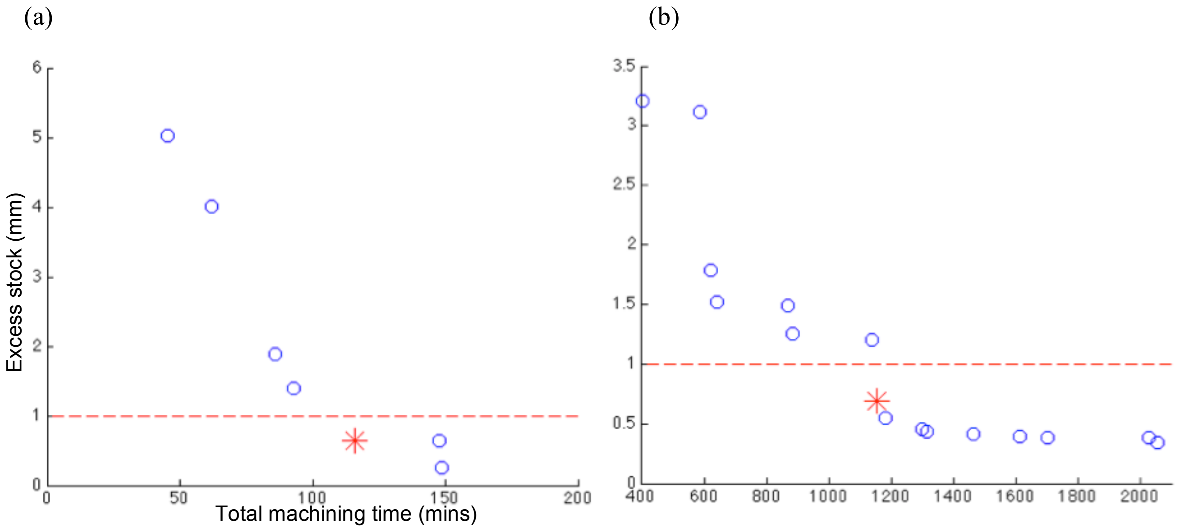
Constraint relaxation through multi-objectivisation has been found to remove search biases created by constraint handling techniques (Coello Coello and Lamont, 2004; Watanabe and Sakakibara, 2005). Multi-objectivisation also can avoid local optima by following multiple search directions (Knowles et al., 2001) and encourage diversity in the population. (Lehman et al., 2013) showed that it can provide increased diversity that can improve performance in a deceptive domain.

A potential problem with the constraint relaxation approach is that search may not follow the direction desired, as infeasible solutions can become Pareto optimal, and multi-objective search may spend a large amount of time in these regions (Runarsson and Yao, 2005). This can happen in the Tool Selection Problem as a solution that leaves a large amount of excess stock may (and perhaps is likely to) have a shorter machining time, making it Pareto optimal. It has also been shown that constraint relaxation with a multi-objective approach can be hampered by not including a search bias towards the feasible region (Runarsson and Yao, 2005). Work in the previous chapter showed that incorporating guided search through the Guided Elitism algorithm improved the effectiveness of NSGA-II on the Knapsack Problem. In this chapter preferential multi-objective search techniques are also tested to see if they improve the performance in finding the minimum value of $f_{eval}(\boldsymbol{E}, p)$.

7.2.2 The Pareto Front of the Tool Selection Problem

The Pareto front created by minimising f_m and f_d provides very useful information to a machinist. It shows the fastest tool sequence that can achieve a certain surface tolerance. This allows the decision maker to see if there are solutions that, for example, have a slightly longer machining time but considerably lower surface tolerance. We will first look at the Pareto fronts for components under investigation in this chapter (parts 1 and 4) and then discuss methods for encouraging convergence on desirable solutions and issues with small population sizes.

Figure 7.1. The Pareto front for (a) Part 1 and (b) the hybrid Part 2-4-H. The x-axis shows total manufacturing time in minutes and the y-axis shows excess stock in mm. The solution that is optimal in the single objective formulation of the problem, and considered to be the optimum here, is marked with a *. The dotted line separates solutions under 1mm.



7.2.2.1 The Pareto Fronts

Figure 7.1 shows the Pareto front, the non-dominated tool sequences with respect to equations 7.3 and 7.4, found on Part 1 and Part 2-4-H, which are the parts under investigation in this chapter. The red star shows the solution that is optimal in the single objective formulations of the problem (according to f_{eval} from 7.2), the solution with the fastest machining time with excess stock less than 1mm in all places. This was found by analysing the cached search spaces for both parts. The figure shows that Part 2-4-H has a more complex Pareto front than Part 1. There are 16 non-dominated solutions, compared to just 7 for Part 1.

Both of the fronts displayed in Figure 7.1 show that multi-objectivising the Tool Selection Problem by considering excess stock as an independent objective creates many non-dominated solutions that are undesirable to us. These are Pareto optimal solutions that have short machining times but large amounts of remaining excess material. As discussed in the previous chapter, multi-objective algorithms such as NSGA-II attempt to return an even spread of solutions along the Pareto front. Returning an approximation of the Pareto front is a useful feature, as it provides a set of alternative solutions. For example, with Part 2-4-H there is a solution that has a similar machining time to the starred solution that can achieve a lower surface tolerance. With Part 1, there is a solution with a longer machining time but a considerably lower surface tolerance. These are both interesting alternative solutions for a process planner to consider. With no preferences incorporated into the algorithms, they may not return the specific solution that the decision maker deems most important. Next we will discuss methods that will be used in this chapter to encourage convergence on the solution we want and issues with obtaining the Pareto front when using small population sizes.

7.2.2.2 Selecting Solutions on the Front

Figure 7.1 shows that for both parts there are several ‘incomparable’ non-dominated solutions present in the Pareto front. The Tool Selection Problem instance is discrete and so there are a finite number of solutions. For a non-archiving algorithm like NSGA-II, the number of solutions returned is entirely related to the population size. If we have a population size, N, smaller than the Pareto front, the greatest number of Pareto optimal solutions that can be returned is N. In a multi-objectivisation problem, multi-objective techniques are being used but the goal is still to find a specific solution.

In order to make sure that the correct solution is found, intuitively one might decide to set the population to the size of the Pareto front. However, it is difficult to know what this is without prior knowledge of the search space. This is particularly difficult when there is a large computational cost associated with evaluations. Small population sizes can take less time to converge (Ceroni et al., 2001), which can be an advantage when dealing with expensive evaluations, providing that there is enough diversity present to reach good solutions. The results from Chapter 5 showed that using a small population size provided a good balance between high quality solutions and low numbers of evaluations. However, this tactic will naturally restrict the number of Pareto optimal solutions that are returned.

In the multi-objectivisation case we assign considerable importance to finding the optimal solution from the perspective of the single-objective fitness function. Therefore, in the experiments in this chapter, special emphasis is placed on concentrating algorithms on the interesting region of the search space, even when using small populations. To accomplish this, preferential search strategies are applied using the R-NSGA-II algorithm (Deb and Sundar, 2006) and the Guided Elitism adaptation to NSGA-II introduced in the previous chapter. We also introduce an amendment to the crowding distance assignment in NSGA-II, to encourage more diversity when used with smaller population sizes. The best solution discovered according to the single objective formulation, i.e. the solution that minimises $f(x)$ in eq. (1), at each evolutionary cycle is also archived, in a passive manner.

7.3 Methods

In this section we will first describe the experiment and then introduce the exact implementation of the five algorithms used in this chapter – GA, RRSHC, NSGA-II, NSGA-II with Guided Elitism (GE) and R-NSGA-II. We will also introduce a modification of the Crowding Distance Assignment method in NSGA-II and explain the motivation behind it.

7.3.1 Experiment

7.3.1.1 The Task

The task in this experiment is the same as in Chapter 5, to find a sequence of up to 5 tools that, in the shortest amount of time, can machine a component so that the maximum thickness of excess material (stock) that deviates from the desired final shape in any one place is less than 1mm. Tools are chosen from the same library of 18, as seen in Table 4.1 from Chapter 4. We compare the performance of the algorithms on two components. The first is Part 1, which was shown to have two local optima, when applying a Steepest Ascent Hill Climbing algorithm. This component was chosen to see if the multi-objective algorithms performed similarly to the single-objective metaheuristics, which were shown to be successful on this part. The second component selected is Part 2-4-H (as presented in Chapter 5), chosen because conversely it has several local optima and a difficult search landscape. The single-objective algorithms found it very difficult to consistently locate the optimal solution in this environment.

7.3.1.2 Evaluations and Budgets

Again, identically to Chapter 5, tool sequences are evaluated using Vero Software's Machining Strategist CAM software, which gives a full tool path for each tool and a 3D representation of the final surface of the machined part. A cached version of the search space is used, to allow for the rigorous testing of the algorithms.

The emphasis of these experiments differs slightly from Chapter 5. Here, the goal is to discover which algorithms are most successful at reaching the optimum solution, as opposed to finding algorithms that strike a good balance between evaluations and search success. In order to compare the algorithms fairly, and also to indicate how well they can be used in real world time-constrained situations, they are tested with different forced evaluation budgets.

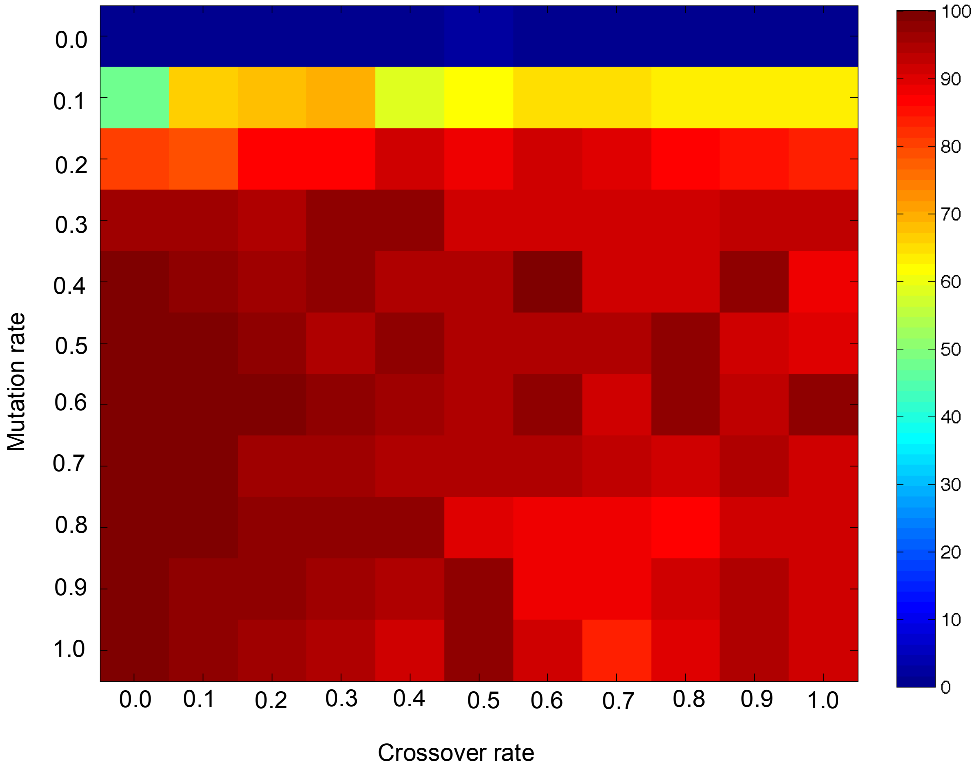
7.3.2 Algorithms

Three main algorithms are tested on the task described above. These are the single-objective Genetic Algorithm (GA), Random Restart Stochastic Hill Climbing (RRSHC), and the Elitist Non Dominated Sorting Genetic Algorithm II (NSGA-II). Two variations of NSGA-II are tested that included preferences in the search. These are the established Reference Point NSGA-II (R-NSGA-II) and the Guided Elitism NSGA-II (GE). In addition, the Crowding Distance Assignment in NSGA-II has been modified to prevent certain duplicate solutions being given an infinitely high diversity measure. This version is denoted as NSGA-II*. All of the algorithms use the same methods for representation, mutation and recombination, the latter of which is not used in RRSHC. The genetic operators and representation are exactly the same as those described in Chapter 5, with the Gradual Mutation operator being used.

7.3.2.1 Genetic Algorithm

The same version of the single-objective GA used in the experiments in Chapter 5 is employed. The algorithm uses single solution elitism and rank selection. The same mutation and crossover rates are used, of 0.4 and 0.7 respectively. However, in this experiment many different population sizes are tested. The stopping criterion is also different, now using a hard evaluation limit, with different values assessed. To evaluate solutions, the single-objective fitness function, f_{eval}, is used, as described in equation 7.2 above. This is the total machining time with a punishment factor for solutions with a maximum excess stock above 1mm.

Figure 7.2. Results for the parameter sweep for NSGA-II with a population size of 15. The colours indicate the number of (single-objective) optimal solutions found on Part 2-4-H over 100 trials with a 500 evaluations budget.



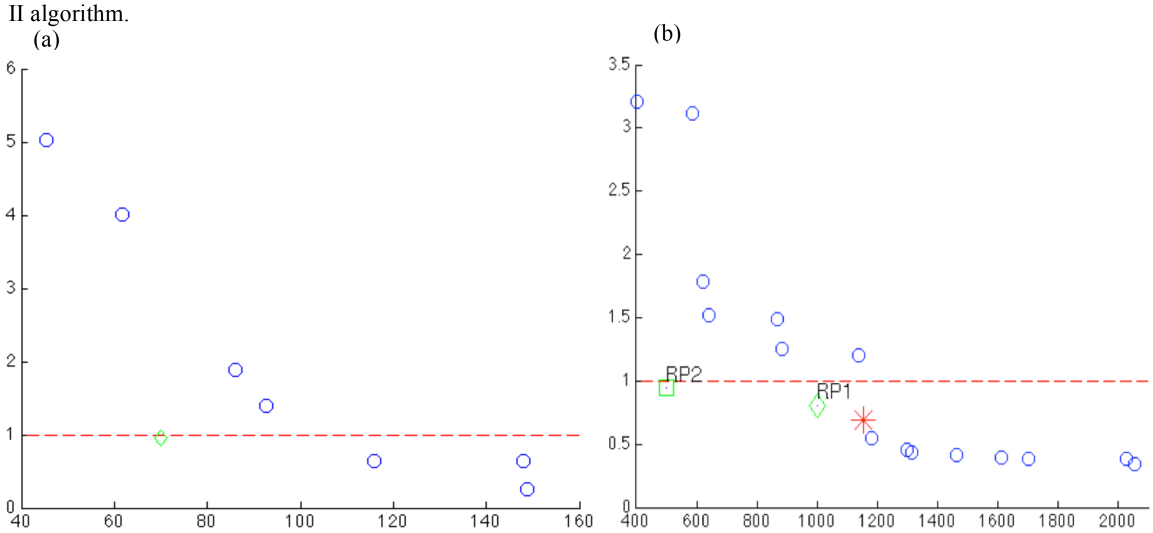
7.3.2.2 Random Restart Stochastic Hill Climbing

Again, a similar version of the RRSHC algorithm is used as in the experiments in Chapter 5. The one change here is that rather than having a set number of restarts, a new Hill Climbing run begins when the number of evaluations in the current run reaches a certain limit, with many different values tested in the experiments presented later in this chapter. The search ends when the total cumulative number of evaluations reaches a predefined limit. As with the GA, the single-objective fitness function, f_{eval}, is used, as described in equation 7.2.

7.3.2.3 NSGA-II

The NSGA-II algorithm introduced in the previous chapter is used as the base multi-objective algorithm. The same genetic operators and representation is implemented as in the GA. Many different permutations for the mutation and crossover rates were tested over 100 runs using a population of 15 on Part 2-4-H, with a budget of 500 evaluations. The results from this are shown in Figure 7.2. It should be noted that with mutation rates higher than 0.2, there does not seem to be much sensitivity to the parameters. It was decided to set a mutation rate of 1.0 and a crossover rate of 0.1. These same conditions were used for all of the algorithms. Although this mutation rate may seem high, only one small change is made to the child, which occurs when the offspring is generated. As we keep the original population, without this mutation rate there would be a large number of duplicates in the child population. Two objective functions are used, as

Figure 7.3. Plot showing the Pareto optimal fronts for (a) Part 1, with the reference point in a green diamond and (b) Part 2-4-H, with the reference points RP1 (diamond) and RP2 (square). The reference points are used by the R-NSGA-II algorithm.



described by equations 7.3 and 7.4. These are, f_m, the total machining time, and f_d, the maximum distance of excess stock.

7.3.2.4 R-NSGA-II

R-NSGA-II was also introduced in the previous chapter. It is a method used for providing preferences in multi-objective search, and was seen to work well in both a discrete and continuous environment. The decision maker (DM) sets a point in objective space and solutions should be returned that are close to this region, giving the DM a set of solutions more relevant to their search goal. The algorithm modifies the crowding distance assignment in NSGA-II, so that solutions are ranked based on how far away they are from their closest reference point, rather than their distance from each other. In this way, solutions with the same Pareto ranking are treated more favourably if they are closer to the DM defined reference point.

The algorithm is set up in the same way as NSGA-II, using the same parameter values, representation and genetic operators. The same two objective functions are used, f_m and f_d, as described by equations 7.3 and 7.4. Additionally, the epsilon parameter is set to 0.001mm. The single reference point used on Part 1 and the two reference points used on Part 2-4-H (labelled RP1 and RP2) can be seen Figure 7.3.

7.3.2.5 Guided Elitism

Guided Elitism (GE) was introduced in the previous chapter, where it was successfully applied to the 0/1 Knapsack Problem. The idea behind GE is to combine the free unconstrained NSGA-II with a small amount of extra selective pressure towards the desired region of search space, to encourage preferred results to be returned, while preventing the search from getting stuck in local optima. In the context of the exploration/exploitation paradigm, GE combines the exploration power of the multi-objective algorithms with the exploitation power of weighted aggregate functions. In contrast to R-NSGA-II, it can also protect some dominated solutions, which can help prevent convergence on sub-optimal fronts (Deb and Goel, 2001).

GE and R-NSGA-II both need extra information indicating which region of objective space to head towards. As we are multi-objectivising a problem from a single-objective implementation, the original single-objective fitness function can be used to guide search. This is f_{eval} from equation 7.2, the same fitness function used in the GA and RRSHC implementations. This highlights a difference between GE and R-NSGA-II, that in GE you do not need to necessarily have advance knowledge of objective space.

On the multi-objective side of the algorithm, the same two objectives are used as in the NSGA-II and R-NSGA-II implementations. These are f_m and f_d from equations 7.3 and 7.4. The same representation and genetic operators are used as in the GA, NSGA-II and R-NSGA-II, as well as the same mutation rate of 1.0 and crossover rate of 0.1. GE has an additional parameter, *y*, which determines how many members of the population to protect. This affects the balance between exploration and exploitation, similarly to mutation and crossover rates. Here, we decided to employ only a small amount of guidance and set *y* using,

$$y = round(\frac{population\ size}{10}) \qquad (7.5)$$

7.3.2.6 Modification to NSGA-II's Crowding Distance Assignment

In the Crowding Distance Assignment method in NSGA-II, described in Chapter 6, for each objective the highest and lowest solutions have a crowding distance score (CD) set to infinity. For these solutions, the crowding distance will remain infinite no matter what their other objective values are. In the original algorithm there is no explicit sorting between duplicate solutions, which are considered here to be solutions with identical objective values. This can lead to a case where non-dominated duplicate solutions that score best in one objective and worst in another can both have their crowding distance set to infinity, meaning that they are guaranteed to remain in the population. These duplicate solutions will be chosen over any less crowded dominant solutions created in the child population.

An illustration of this situation is shown in Figure 7.4**,** an example run of NSGA-II applied to Part 2-4-H with a population of 6. Solutions 0 and 1 are duplicates, as are 4 and 5. This gives 4 unique solutions from the six members of the population. As no explicit method is specified for sorting, duplicate solutions can be assigned an arbitrary ordering. For example, using the *sorted()* method from Python's Standard Library will give the ordering seen in Figure 7.4(a). To correct this, we propose an additional procedure that ensures

Figure 7.4. Tables showing in grey shading which members of the population, marked by their id, are set to infinity when using (a) NSGA-II's Crowding Distance Assignment and (b) the modification presented in 7.3.2.6. $\boldsymbol{CD_{f1}}$refers to the crowding distance assigned to a solution after sorting by $\boldsymbol{f_1(x)}$ and $\boldsymbol{CD_{f2}}$refers to the crowding distance assigned to a solution after sorting by $\boldsymbol{f_1(x)}$.

(a)

Id	$f_1(x)$	$f_2(x)$
0	1951.4	0.4
1	1951.4	0.4
2	641.7	1.5
3	1395.1	1.3
4	403.7	3.2
5	403.7	3.2

Unsorted Population

Id	$f_1(x)$
4	403.7
5	403.7
2	641.7
3	1395.1
0	1951.4
1	1951.4

Sorted by $f_1(x)$

Id	$f_2(x)$
0	0.4
1	0.4
3	1.3
2	1.5
4	3.2
5	3.2

Sorted by $f_2(x)$

Id	$f_1(x)$	CD_{f2}	CD_{f1}
4	403.7	0.0	∞
5	403.7	0.0	238.0
2	641.7	0.0	991.4
3	1395.1	0.0	1309.7
0	1951.4	0.0	556.3
1	1951.4	0.0	∞

Sorted by $f_1(x)$

Id	$f_2(x)$	CD_{f2}	CD_{f1}
0	0.4	556.3	∞
1	0.4	∞	∞
3	1.3	1309.7	1310.6
2	1.5	991.4	993.3
4	3.2	∞	∞
5	3.2	238.0	∞

Sorted by $f_2(x)$

(b)

Id	$f_1(x)$	$f_2(x)$
0	1951.4	0.4
1	1951.4	0.4
2	641.7	1.5
3	1395.1	1.3
4	403.7	3.2
5	403.7	3.2

Unsorted Population

Id	$f_1(x)$
4	403.7
5	403.7
2	641.7
3	1395.1
0	1951.4
1	1951.4

Sorted by $f_1(x)$

Id	$f_2(x)$
0	0.4
1	0.4
3	1.3
2	1.5
4	3.2
5	3.2

Sorted by $f_2(x)$

Id	$f_1(x)$	CD_{f2}	CD_{f1}
4	403.7	0.0	∞
5	403.7	0.0	238.0
2	641.7	0.0	991.4
3	1395.1	0.0	1309.7
0	1951.4	0.0	556.3
1	1951.4	0.0	∞

Sorted by $f_1(x)$

Id	$f_2(x)$	CD_{f2}	CD_{f1}
1	0.4	∞	∞
0	0.4	556.3	557.2
3	1.3	1309.7	1310.6
2	1.5	991.4	993.3
5	3.2	238.0	239.7
4	3.2	∞	∞

Sorted primarily by $f_2(x)$ and secondarily by CD_{f2}

Algorithm 7.1: Pseudo-Code for the crowding distance modification in NSGA-II*

```
initialise the crowding distance, $cd$, to 0 for each solution
for each objective, do
    normalise the objective
    sort the solutions in the Pareto front according to their value for this
    objective, giving sorted_front
    sort any duplicate solutions at the top of sorted_front by the existing $cd$ ascending
    sort any duplicate solutions at the bottom of sorted_front by the existing $cd$ descending
    set $cd$ for the first and last solutions to infinity
    for solutions from solutions[start + 1] to solutions[end – 1], do
        subtract the objective value to the left of the solution from the value to the right, and
        add this value to the solution's $cd$
    end for
end for
```

that only one of the duplicate solutions is given an infinite crowding distance, which is described in the pseudo-code presented in Algorithm 7.1. For the solutions in a Pareto front, the group is sorted by each objective value in turn. Any duplicate solutions at the start and end of the sorted group are then resorted by their current crowding distance. For example, in Figure 7.4(a) where solutions 0, 1, 4 and 5 would all be given infinite crowding distance normally, our procedure assigns this value only to solutions 0 and 5, which is seen in Figure 7.4(b). This adaptation to the algorithm is referred to as NSGA-II*. Experiments use the same parameters as the NSGA-II, a mutation rate of 1.0 and a crossover rate of 0.1. The objective functions are also identical, f_m and f_d from equations 7.3 and 7.4.

7.4 Results

In this section we present the results of experiments on two components, Part 1 and Part 2-4-H. Single-objective algorithms were shown to be successful on Part 1 but found the search space of Part 2-4-H much more difficult. The goal of the two experiments is to determine whether multi-objective techniques applied to a single-objective Tool Selection Problem can achieve similar performance when there are a small number of local optima and improved performance when there are large numbers present in the search space, subject to strict evaluation budgets.

A Genetic Algorithm (GA), NSGA-II (N), crowding-distance modified NSGA-II (N*), NSGA-II with Guided Elitism (GE) and R-NSGA-II (R) are all tested on the two components. Additionally, Random Restart Stochastic Hill Climbing (RRSHC) is applied to Part 2-4-H because there are more local optima and we wanted to compare the performance of a restart algorithm with the multi-objective techniques. The algorithms are all assessed over 1,000 trials in order to evaluate their consistency. They are evaluated solely on their ability to find the optimal solution from a single-objective perspective, the solution with the fastest machining time and excess material under 1mm in all places. This will be referred to as the optimum throughout the rest of this section. The algorithms are run for different evaluation budgets, which will be notated as *c*-EL, where *c* is the maximum number of unique evaluations allowed.

7.4.1 Optimising Tool Selection for Part 1

On this component the algorithms are tested with 15 different population sizes, ranging from 5 – 20 in increments of 1. These population sizes were chosen based on the tight evaluation budgets, and the success of the GA with a population size of 9 in the experiments in Chapter 5. Four evaluation budgets are tested, 100, 150, 200 and 250, also based on the performance of the single-objective metaheuristics from Chapter 5.

The results for this component are very interesting. Many configurations of the single-objective GA outperform the multi-objective algorithms, given these evaluation limits. The boxplots in Figure 7.5 show the distribution of optimum solutions obtained by the 15 different population size configurations of the algorithms, over 1,000 trials. The Run Length Distribution plot in Figure 7.6 show the top 25th percentile of optimum solutions obtained by the algorithms, for different evaluation limits. From these two figures we can see that the GA has very high median and 3rd quartile scores. There is much less variability among different population size configurations with the GA, especially from 150-*EL* and above. This is reflected in Table 7.1, where we can see that the GA has the smallest interquartile range across all four evaluation limits. Figure 7.7 shows the number of optimal solutions found for each population size across the evaluation budgets. This shows that the GA is much more robust when it comes to population size changes and performs particularly well with small population sizes.

Among the multi-objective algorithms, R is the best performing at the lowest evaluation limit (100-*EL*), displaying the highest minimum, median and maximum scores. Looking at Figure 7.7, it performs well with very small population sizes, which is reflected in low interquartile ranges. It also achieves the highest score (equal on 200-*EL* and 250-*EL*) on three of the four evaluation limits. Even at the higher evaluation limits, the multi-objective algorithms have a lot of variability in the number of optimal solutions found, among the different population sizes. N*, which has a crowding distance modification, also performs relatively well with small population sizes, giving it a much smaller interquartile range compared to N. It has a very similar overall performance to N, reflected in similar median and 3rd quartile score.

N and GE have much lower minimum scores than N* and R. Looking at Figure 7.7, N and GE perform particularly badly with small population sizes. N* and R perform much better with smaller populations, which suggests that the Crowding Distance Assignment in NSGA-II can hamper performance with small populations. GE has by far the most varied behaviour, and is the most sensitive to changes in population size. It does not perform well with small populations, worse than N, which could suggest that the GE modification reduces diversity. Overall, the algorithm does not perform badly on this task, as by 250-*EL* most configurations achieve near-perfect or perfect scores. It also has very strongly performing individual configurations, with Table 7.1 showing that GE has a better maximum score than the GA at every evaluation limit.

The GA's best configuration finds the optimum 81% of the time, while GE achieves 89% and R, 93%. This could provide some evidence towards guided search being effective under strict evaluations budgets, although both GE and R have a lot of variation among different configurations. Interestingly, the non-guided, unconstrained NSGA-II algorithm performs well on this part, which could be due to the small size of the Pareto front.

In summary, on this component the GA performs consistently well across all population sizes and evaluation limits. With certain configurations, the multi-objective algorithms are able to achieve higher numbers of optimal solutions than the GA at all evaluation limits. The multi-objective algorithms are much more sensitive to changes in population size, especially GE and N which use the original NSGA-II Crowding Distance Assignment method. It should be noted that across the 15 population sizes, there are only significant differences between the distributions of the algorithms on 100-*EL* and 150-*EL*, as indicated by the Kruskal-Wallis one way analysis of variance by ranks. Post-hoc analysis using the Wilcoxon Rank Sum test with the Bonferroni correction for multiple comparisons shows that on 100-EL, the GA is significantly different from N, N* and GE, while R is also significantly different from GE ($p < 0.01$). On 150-EL, the only significant difference is the GA compared to GE ($p < 0.01$). This suggests that after 150-*EL*, all of the algorithms perform well on this part.

Figure 7.5. Box plots showing the median radius of catchment areas for all 15 configurations of the 5 algorithms on Part 1 (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The y-axis displays the number of times that the optimum solution was found out of 1,000 runs for each algorithm.

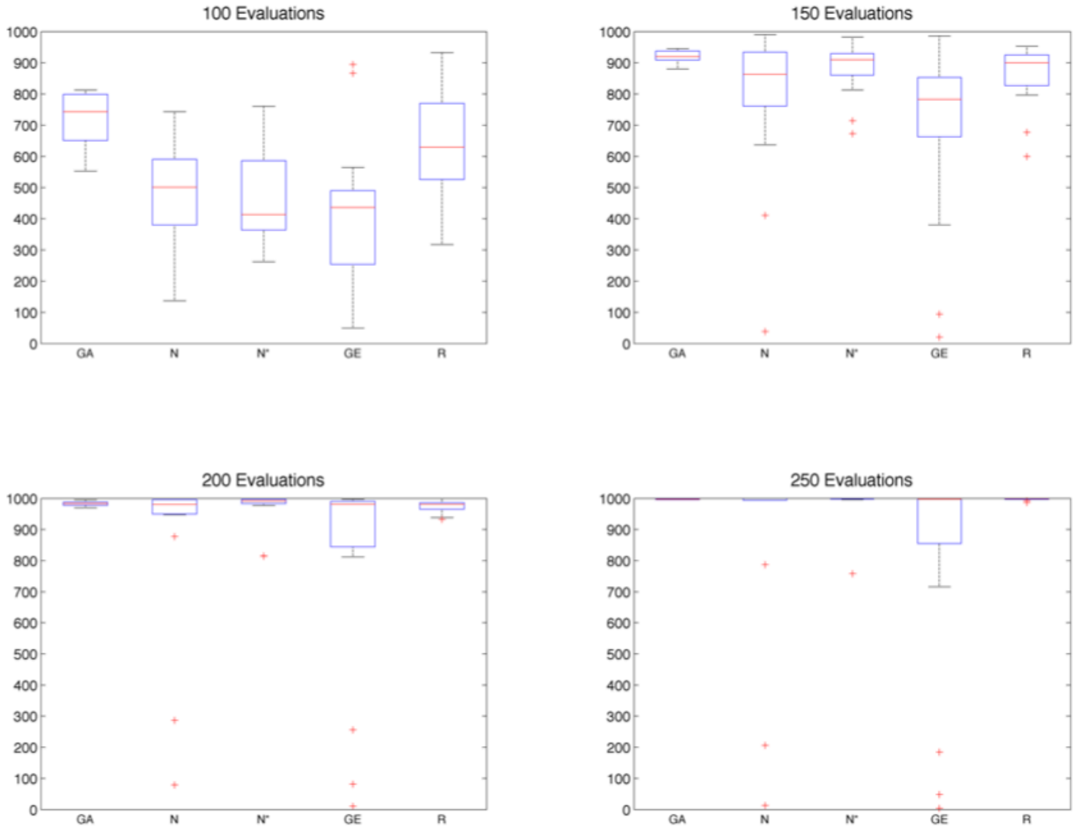


Figure 7.6. Run length distribution plot showing the percentage of optimum solutions found by the top 25th percentile configurations of the algorithms labelled in the legend, after the number of evaluations indicated on the x-axis, applied to Part 1 over 1,000 runs.

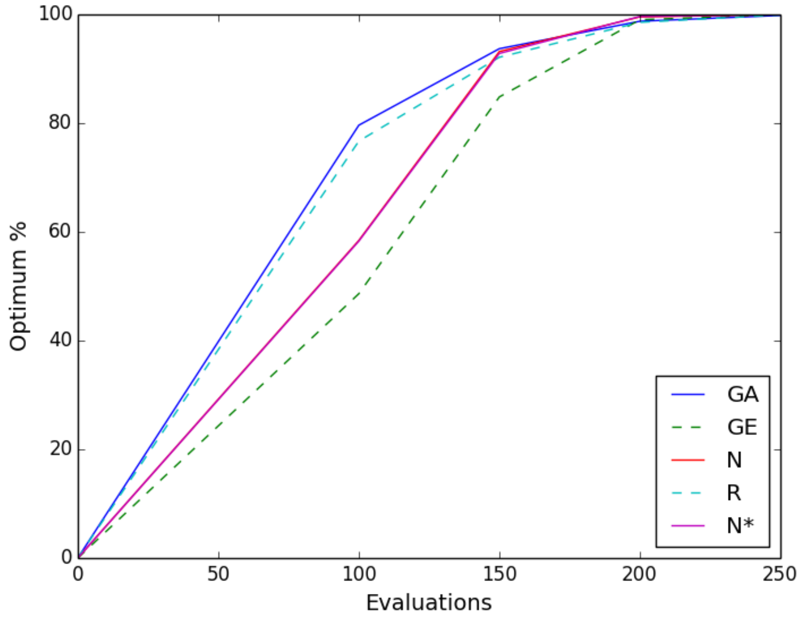


Figure 7.7. Bar plots showing the number of optimum solutions on the y-axis and the population size on the x-axis for each algorithm applied to Part 1.

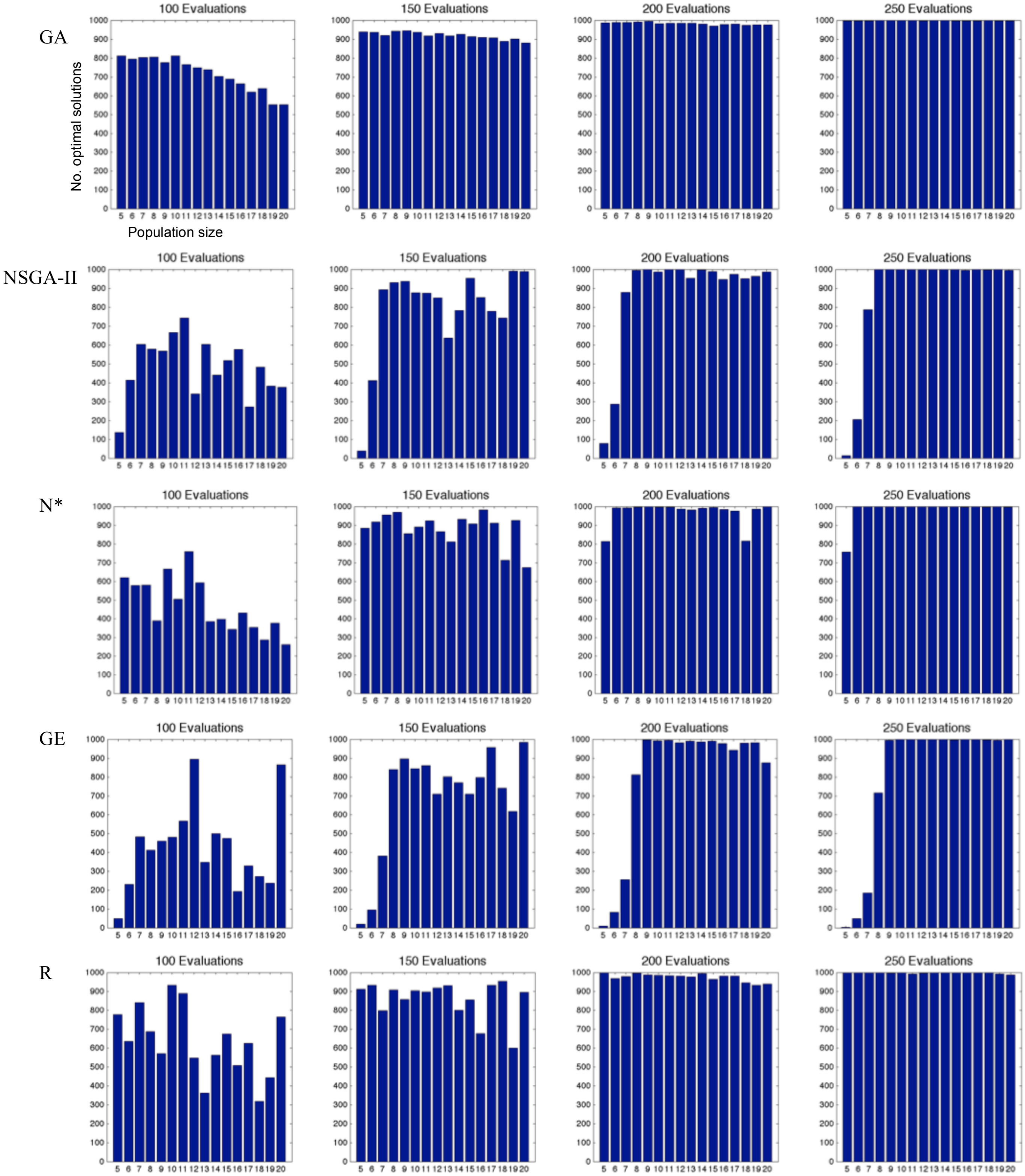


Table 7.1. Showing results from 15 population size configurations on Part 1, over 1,000 trials by the Genetic Algorithm (GA), NSGA-II (N), crowding-distance modified NSGA-II (N*), NSGA-II with Guided Elitism (GE) and R-NSGA-II (R). These are shown for (a) 100, (b) 150, (c) 200 and (d) 250 evaluations. Algorithms in the brackets next to an algorithm name are significantly different ($p < 0.01$) according to a Wilcoxan rank sum test with the Bonferroni correction.

(a) 100 Evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	Interquartile range
GA$^{\mathbf{(N,N*,GE)}}$	553	657	743	796.25	812	139.25
N$^{\mathbf{(GA)}}$	137	381.5	500.5	584.25	743	202.75
N*$^{\mathbf{(GA)}}$	262	370.25	413.5	583	760	212.75
GE$^{\mathbf{(GA,R)}}$	49	263.25	436	486.25	894	223
R$^{\mathbf{(GE)}}$	317	536.25	629.5	767	932	230.75

(b) 150 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	Interquartile range
GA$^{\mathbf{(GE)}}$	880	909.25	919	937	945	27.75
N	38	770	862.5	931.75	990	161.75
N*	673	862.5	909.5	928.25	982	65.75
GE$^{\mathbf{(GA)}}$	20	686	782.5	848.25	985	162.25
R	600	841.5	899.5	921.25	953	79.75

(c) 200 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	Interquartile range
GA	970	978.5	983.5	987.5	995	9
N	79	950.75	981	995.5	1000	44.75
N*	814	983.75	991.5	996.25	1000	12.5
GE	10	860	981.5	990.25	997	130.25
R	933	966	981	985.5	1000	19.5

(d) 250 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	Interquartile range
GA	996	996	997	998.25	999	2.25
N	13	994	1000	1000	1000	6
N*	758	997.75	1000	1000	1000	2.25
GE	4	924.5	997.5	1000	1000	75.5
R	987	997	999	999.25	1000	2.25

7.4.2 Optimising Tool Selection for Part 2-4-H

On Part 2-4-H the Evolutionary Algorithms (GA, N, N*, GE, RP1 and RP2) were tested with 16 different population sizes from 5 – 15 in increments of 1 and additionally, 20, 25, 30, 35 and 40. The larger population sizes were chosen because in Chapter 5 the GA was shown to perform better with larger population sizes on Part 2-4-H. RRSHC was tested with 16 different hill climbing evaluation limits within a single run, 10 – 160 in increments of 10. R-NSGA-II is tested with two reference points, denoted as RP1 and RP2 because preliminary tests suggested that the choice of reference point affected performance on this part. We will first discuss the results of the single-objective algorithms, GA and RRSHC, before moving on to the multi-objective techniques.

7.4.2.1 Single-objective Algorithms

The boxplots in Figure 7.8 show the distribution of the frequency that the optimum solution was found, from each of the 16 population size configurations that the algorithms were tested with, across the four evaluation limits. Looking first at the single-objective algorithms, it is clear that they compare favourably to the multi-objective techniques at the lowest evaluation limit of 150-*EL* but are greatly outperformed at higher evaluation levels. In terms of the scores from the top 25th percentile seen in the Run Length Distribution plot in Figure 7.9, and the median score from all the configurations shown in Table 7.2, the GA is more successful than RRSHC until 350-*EL* where RRSHC narrowly wins.

At 500-*EL*, a large gap emerges between the two algorithms. The best configuration of RRSHC finds the optimum 540 times compared to 413 by the GA. There is only a small improvement in the GA's search success when moving from 350-*EL* to 500-*EL*. Figure 7.12 shows the number of optimal solutions found for each population size across the evaluation budgets. From this we can see that there does not seem to be much improvement between 350-*EL* and 500-*EL* with the smaller population sizes but there is with the larger ones. This is similar to the results found in Chapter 5 (5.4.4.2), which showed that the GA performed much better on this part with large populations, with the best being 80. However, this came with a high number of evaluations, almost 800, and success rate was still only 62%.

The performance of RRSHC steadily increases as the evaluation limit rises. A similar trend is seen in the results from Chapter 5 (5.4.4.2). The larger the evaluation limit, the more restarts that are possible (when the individual hill climbing run reaches its stopping limit). This enables the algorithm to avoid getting stuck in local optima. As seen in Figure 7.13, across all of the evaluation limits, RRSHC performs badly with a per restart evaluation limit of 10. At 350-*EL* and 500-*EL* restart limits of 20 do not perform as well as the others. Similarly, using a limit of 30 produces a worse performance at 500-EL. This suggests that even though more restarts are possible, the depth of search is not great enough.

The pseudo-colour plots in Figure 7.10 show the number of optimal solutions found for each algorithm as bands of colour. From this we can see that both of the single-objective algorithms are much less sensitive to changes in configurations than the multi-objective implementations, as within each evaluation band there is much less variation in shade. This is also reflected in considerably lower interquartile scores. This means that there is less reliance on choosing optimal parameters, which can be difficult when there is no great advance knowledge of the search space.

7.4.2.2 Multi-objective Algorithms

Moving on to the multi-objective algorithms, we will consider N first. Concentrating on the best configurations, the algorithms achieve a good rate of optimal solutions compared to the single-objective techniques. While similar to the GA at 150-*EL*, success increases greatly with higher evaluations, achieving maximum scores of 56%, 87% and 100% at 250-*EL*, 350-*EL* and 500-*EL* respectively. However, as we also see with the other multi-objective algorithms, there is great variability between the success rates of different population sizes. As can be seen in the blue patches in the pseudo-colour plot in Figure 7.10, N performs particularly badly with small populations. The bar charts in Figure 7.12 show that it does not reach the optimum a single time with a population of 5, regardless of the number of evaluations used. The modified version, N*, improves on the worst scores (i.e. the total machining times with punishment factor) of N and achieves much better median and 3rd quartile scores, with comparable best scores, at all evaluation levels. Looking at Figure 7.10, N* has lighter regions in the smaller population sizes, indicating better performance, and less variation in shade. This is reflected in lower interquartile scores than N.

On average, the GE adaptation produces considerable improvements on the performance of N. Median and 3rd quartile scores are higher than N and N* for all evaluation limits and the best configuration can reach the optimum almost 97% of the time with 350 evaluations compared to 87% for N and 39% for the GA. However, the colour plots and boxplots show that there is a lot of variability among the different population sizes. Similarly to N, GE performs badly with small population sizes, with a consistently worse performance with a population of 5 and 6 than the GA, across all for evaluation limits. Although GE is sensitive to changes in population size, it has a smaller interquartile score than N at each limit.

Two reference points were tested for the R-NSGA-II, which are labelled in the figures as RP1 and RP2. The optimum solution has a total machining time of 1,153 minutes, with 0.7mm of excess stock. RP1 uses a reference point very close to this, with 1,000 minutes and 0.8mm of excess stock. RP2 uses an unreachable point, 500 minutes machining time and 0.95mm excess stock. Our intuition was that RP1 would outperform RP2 because it is closer to the optimum solution. However, the results showed the opposite situation. RP2 performs far better than RP1 in terms of maximum, median and 3rd quartile scores. The best configuration of RP2 beats GE at every evaluation limit. However, GE outperforms it in the top 25th percentile, for the higher evaluation limits, largely because RP2 performs badly with large population sizes. Individual configurations of RP2 score very highly, with the best achieving 82% and 99% at 250-*EL* and 350-*EL*. The variability between population sizes, indicated by the interquartile range, is similar to GE and much improved on N for all evaluation limits and N* at 500-*EL*. Figure 7.12 shows that RP2 deals better with smaller population sizes than GE and N, although it displays a consistently bad performance with a population size of 5.

Interestingly, RP1 and RP2 perform the best at the lowest evaluation limit, which could suggest that this a good method to use when on a tight budget. However, at the higher limits RP1 performs worse than all the other multi-objective algorithms. This is especially the case at 250-*El* and 350-*EL*. The only advantage to RP1 seems to be its consistently good performance with smaller populations. This is seen in the colour plot and also in its minimum scores, which are often better than the other multi-objective algorithms. In Figure 7.11 two colour plots are shown where the number of optimal solutions found by RP1 are subtracted from those found by RP2. If we look at the top right of the figure, red areas show populations where RP2 outperforms RP1. This occurs almost everywhere, apart from with a population of 5, where RP2 performs badly, and with a population of 40 at 150 and 250-EL. However, RP1 performs much better as the evaluation limits increase, with large jumps in median scores. This implies that it could perform better if allowed to run for more evaluations.

A similar analysis is shown in the bottom half of Figure 7.11, where GE's scores are subtracted from RP2. In the bottom right of the figure, red patches signify RP2 performing better, blue patches where GE dominates and green patches where they are equal. It appears that GE outperforms RP2 in more places. RP2 performs better than GE with the smallest population sizes but GE outperforms it with larger populations, especially for 250-*EL*, 350-*EL* and 500-*EL*. The bottom left plot in Figure 7.11 shows how much better the algorithm performs. There are few red or blue areas, which means that generally the algorithms do not greatly outperform each other.

As with Part 1, the Kruskal-Wallis test was applied to see for each evaluation limit whether there were significant differences between the number of optimal solutions found by the various algorithms, across the tested population sizes. The test indicated that there were significant differences at every evaluation band. Post-hoc analysis with the Wilcoxan Rank Sum test and using the Bonferroni correction showed that there were several significant differences ($p < 0.01$), which are indicated in Table 7.2. There are not many significant differences present at 150-EL, but from 250-EL the most noticeable trend is that GE and RP2 are significantly different from the single-objective algorithms at the three higher evaluation limits. It should be noted that a sample size of 16 populations is small to compare 7 algorithmic implementations, although this analysis shows that while, for example at 500-EL, all of the multi-objective algorithms can achieve much higher scores with their best configurations than the single-objective algorithms can, there is a fair amount of overlap between the two techniques with the worst performing configurations of the multi-objective algorithms.

Figure 7.8. Box plots showing the median radius of catchment areas for all 16 configurations of the 7 algorithms on Part 2-4-H (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The y-axis displays the number of times that the optimum solution was found out of 1,000 runs for each algorithm.

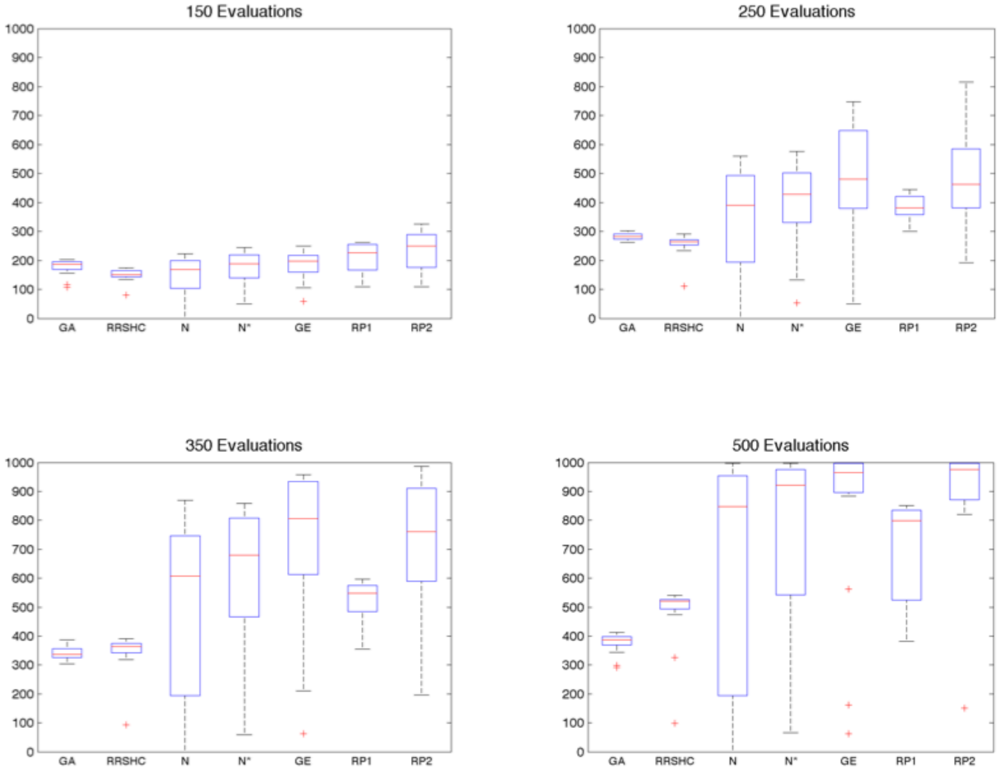


Figure 7.9. Run length distribution plot showing the percentage of optimum solutions found by the top 25th percentile configurations of the algorithms labelled in the legend, after the number of evaluations indicated on the x-axis, applied to Part 2-4-H over 1,000 runs.

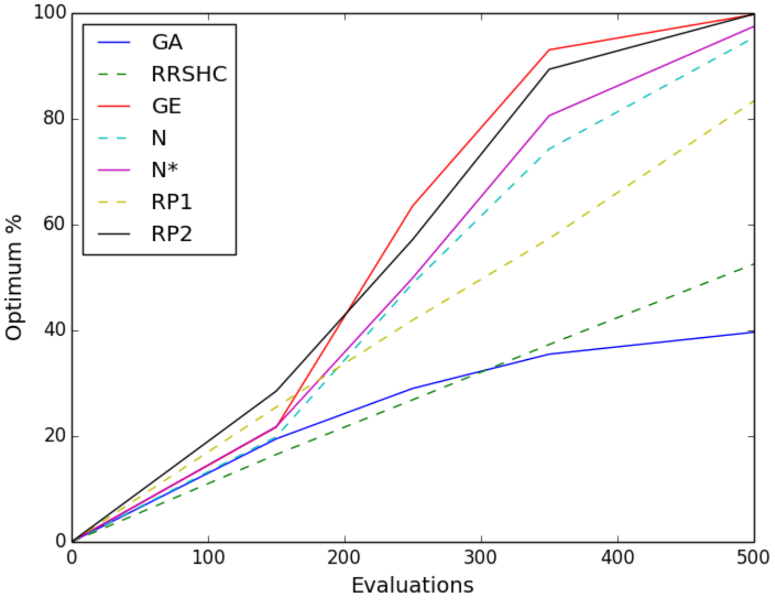


Figure 7.10. Pseudo-colour plots showing for each algorithm on Part 2-4-H, the number of optimum solutions found for each population size (or number of evaluations in a run for RRSHC). Population size is on the y-axis and the evaluations limit is on the x-axis. Dark blue shows low search success, while dark red shows high success.

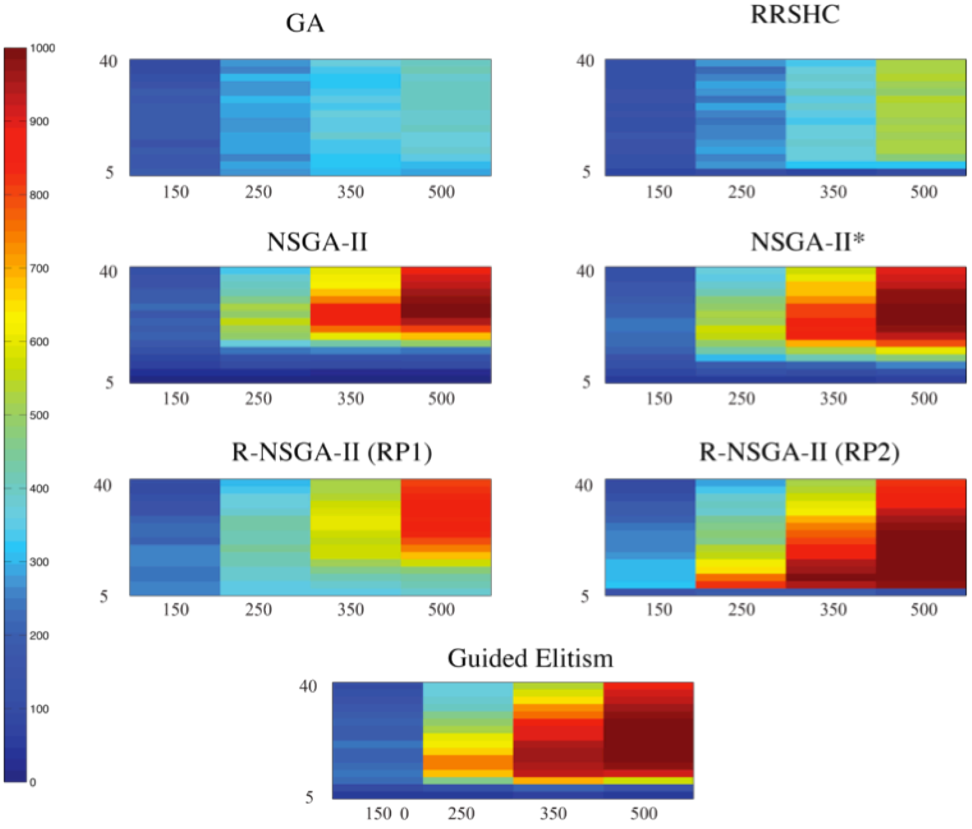


Figure 7.11. Pseudo-colour plots showing on the left, the number of optimum solutions found for each population size by RP2 subtracted by RP1 (top) and GE (bottom), from Part 2-4-H. Population size is on the y-axis and the evaluations limit is on the x-axis. Dark red shows that RP2 performs much better, while dark blue represents a much worse performance. On the right, the plots show for each population size if RP2 is better (dark red), worse (dark blue), or the same (green) as RP1 (top) or GE (bottom).

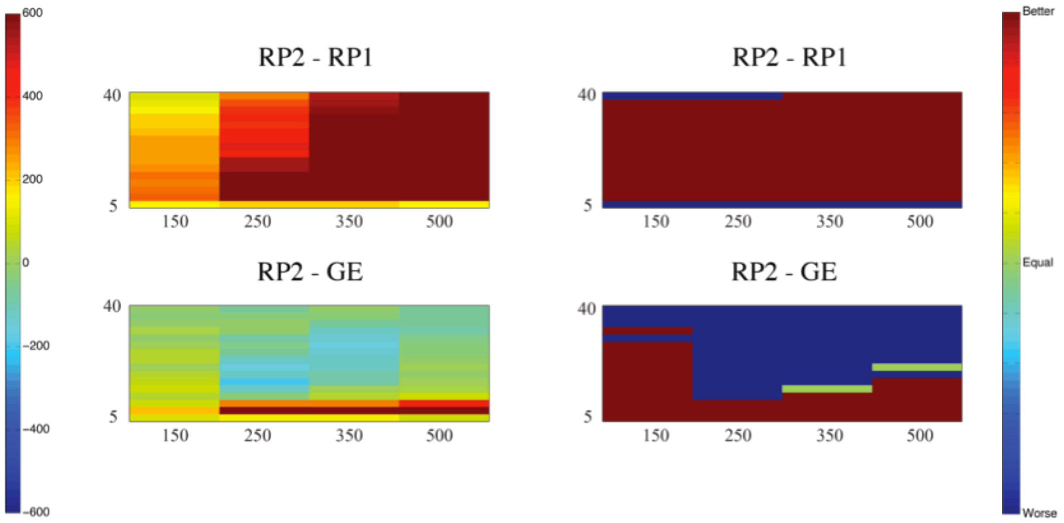


Table 7.2. Showing results from 16 population size configurations on Part 2-4-H, over 1,000 trials by the Genetic Algorithm (GA), NSGA-II (N), crowding-distance modified NSGA-II (N*), NSGA-II with Guided Elitism (GE) and R-NSGA-II (R). These are shown for (a) 100, (b) 150, (c) 200 and (d) 250 evaluations. Algorithms in the brackets next to an algorithm name are significantly different ($p < 0.01$) according to a Wilcoxan rank sum test with the Bonferroni correction.

(a) 150 Evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	IQR
GA	108	169.5	187	195	203	25.5
RRSHC$^{(RP1,RP2)}$	80	144.5	150.5	165.25	173	20.75
N$^{(RP2)}$	0	117.25	168.5	198.75	222	81.5
N*	49	141.75	187.5	218	245	76.25
GE	58	164.75	197.5	217	249	52.25
RP1$^{(RRSHC)}$	109	168.25	226.5	255	263	86.75
RP2$^{(RRSHC,N)}$	109	182.25	249.5	285.25	326	103

(b) 250 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	IQR
GA$^{(GE,RP2)}$	263	273	282.5	290.25	301	17.25
RRSHC$^{(N*,GE,RP1,RP2)}$	110	254.25	264	268.75	291	14.5
N	0	217	390.5	489	560	272
N*$^{(RRSHC)}$	53	342	428.5	499.25	575	157.25
GE$^{(GA,RRSHC)}$	49	382.25	479.5	635.5	747	253.25
RP1$^{(RRSHC)}$	300	359.25	381	419.5	444	60.25
RP2$^{(GA,RRSHC)}$	191	383	463	572	815	189

(c) 350 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	IQR
GA$^{(N*,GE,RP2)}$	304	326	337.5	355	388	29
RRSHC$^{(GE,RP2)}$	93	344	364.5	373.25	391	29.25
N	0	227.75	607	743	869	515.25
N*$^{(GA)}$	58	494.25	680	806	859	311.75
GE$^{(GA,RRSHC)}$	63	628.75	805.5	930.5	957	301.75
RP1	355	499.75	548	573.5	596	73.75
RP2$^{(GA,RRSHC)}$	195	596.5	760.5	893.75	986	297.25

(d) 500 evaluations

Algorithm	Minimum	1st Quartile	Median	3rd Quartile	Maximum	IQR
GA$^{(N*,GE,RP2)}$	292	371.25	387.5	396.5	413	25.25
RRSHC $^{(GE,RP2)}$	98	497.75	520	525.5	540	27.75
N	0	220.25	847	953.75	998	733.5
N*$^{(GA)}$	65	573.5	922	974.75	998	401.25
GE$^{(GA,RRSHC)}$	62	903.5	965	997.25	999	93.75
RP1	381	547	798	834	851	287
RP2$^{(GA,RRSHC)}$	150	872	976.5	998	999	126

Figure 7.12. Bar plots showing the number of optimum solutions on the y-axis and the population size on the x-axis for each algorithm applied to Part 2-4-H.

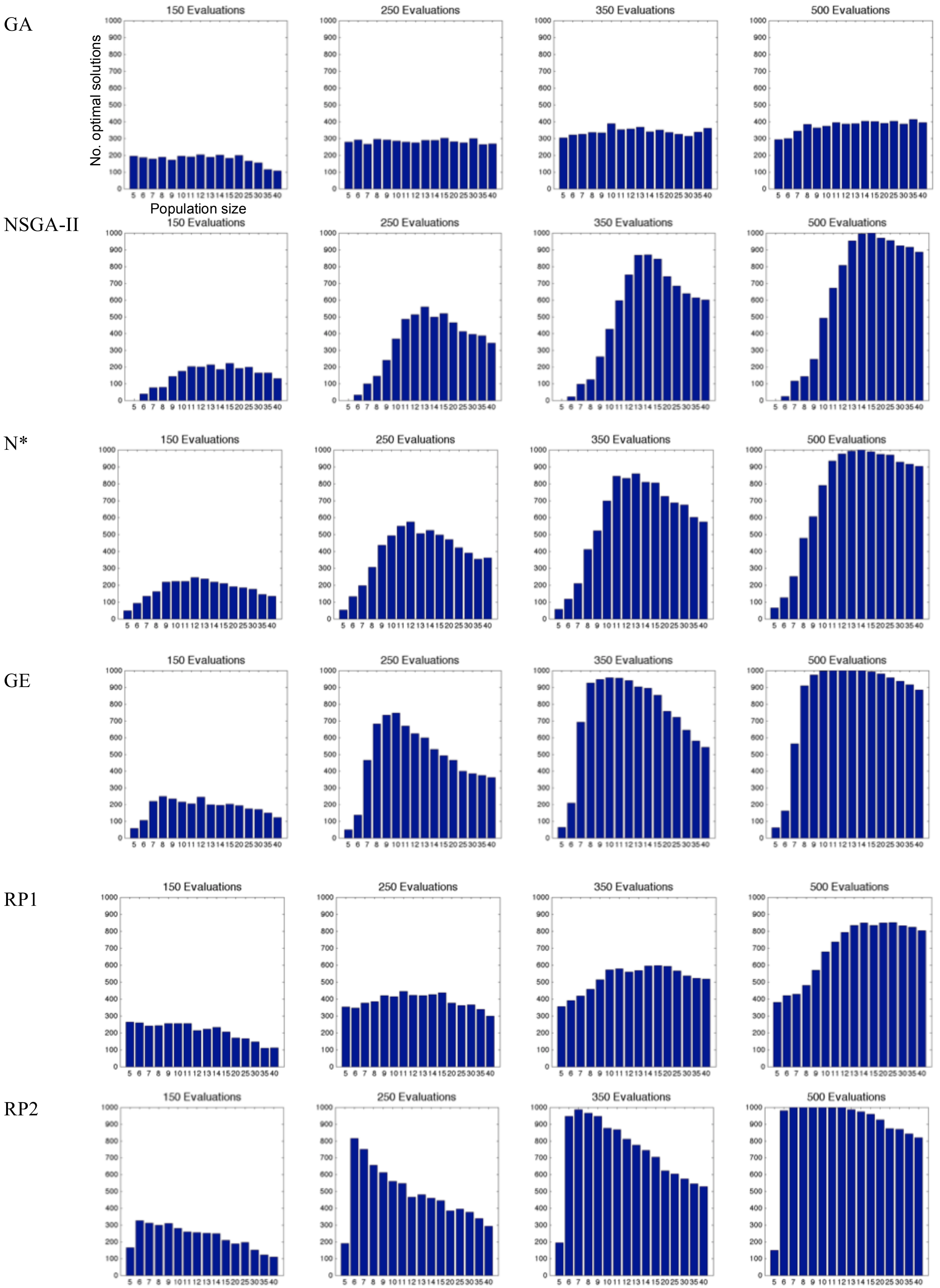
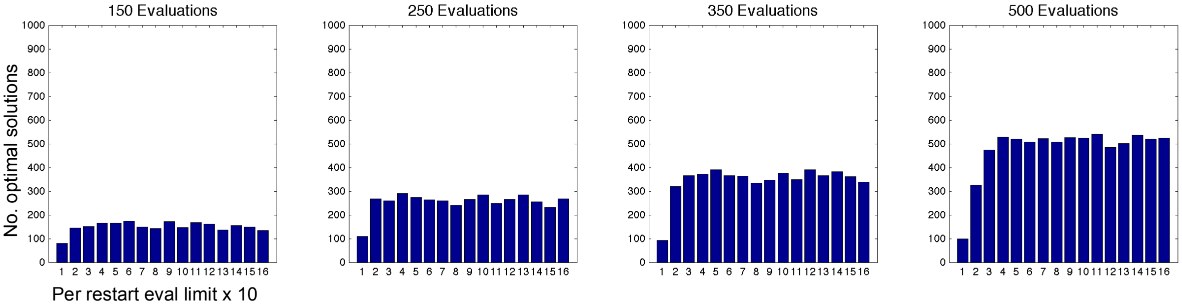


Figure 7.13. Showing for evaluation limits of 150, 250, 350 and 500 the number of optimal solutions (y-axis) found across 1,000 trials for the RRSHC algorithm on Part 2-4-H, for individual Hill Climbing evaluation limits on the x-axis of 10 – 160 in increments of 10



7.4.3 Population Size

The problem of dealing with expensive evaluation functions has been discussed previously in Chapter 5. A successful strategy used in Chapter 5 was to employ small population sizes. GAs that use small populations are often called Micro-Genetic Algorithms (Coello Coello and Toscano, 2001). On Part 1, looking at the number of optimal solutions obtained per population size in Figure 7.7, the GA performs better with smaller population sizes, especially with the lower evaluation limits. However, results for the more difficult component, seen in Figure 7.12 show that this tactic does not make such a big difference, likely due to the GA getting frequently trapped in suboptimal areas. In other words, on this component the fast convergence offered by the small population sizes encourages convergence on suboptimal solutions. Especially at 500-*EL*, we see that the GA's performance improves as population size increases, which suggests that the added diversity improves search success.

It is clear from Figure 7.7 and Figure 7.12 that the multi-objective algorithms are very sensitive to population sizes. Concentrating on Part 2-4-H, NSGA-II and NSGA-II* both display similar behaviour. Search performance is worst with the lowest population sizes and improves as population size increases until it peaks and falls off with the largest sizes. As these algorithms are working without enforced constraints or guidance, the population size has to be reasonably large in order to make sure that solutions in the preferred regions of the Pareto front are returned. NSGA-II*, with increased diversity protection, works better with smaller population sizes and has better median scores in most cases on both parts.

Guided Elitism performs badly with very small population sizes but returns consistently good results when utilised with at least 8 members. Increasing population size improves performance before it drops off with the larger sizes. The size of the population before the drop off increases with evaluation budget. This implies that GE is able to converge quickly with smaller population sizes but needs more evaluations to converge with larger populations. Both NSGA-II and Guided Elitism perform extremely badly with small population sizes compared to NSGA-II* and R-NSGA-II, which suggests that the Crowding Distance Assignment in NSGA-II prevents good exploration of the search space with small population sizes.

Figure 7.12 shows interesting behavioural differences between RP1 and RP2. From 250 evaluations, similarly to NSGA-II, NSGA-II* and Guided Elitism, RP1's performance increases with population size before plateauing with the largest sizes. RP2 performs very badly with 5 members but increases dramatically with population size before tailing off sharply with the larger populations. Similarly to GE, it is able to converge quickly on the optimal solution with small population sizes, which is a likely reason for its good performance under strict evaluation budgets. From the bar plots it seems that RP1 needs more evaluations to converge fully on the optimal solution. The similar behaviour of RP1 to NSGA-II could suggest that the search is not getting directed quickly and correctly towards the optimal solution. The good performance of RP2 and GE suggests that NSGA-II can benefit significantly from guidance and converge quickly on optimal solutions with small population sizes, which can create a successful search strategy under tight evaluation budgets.

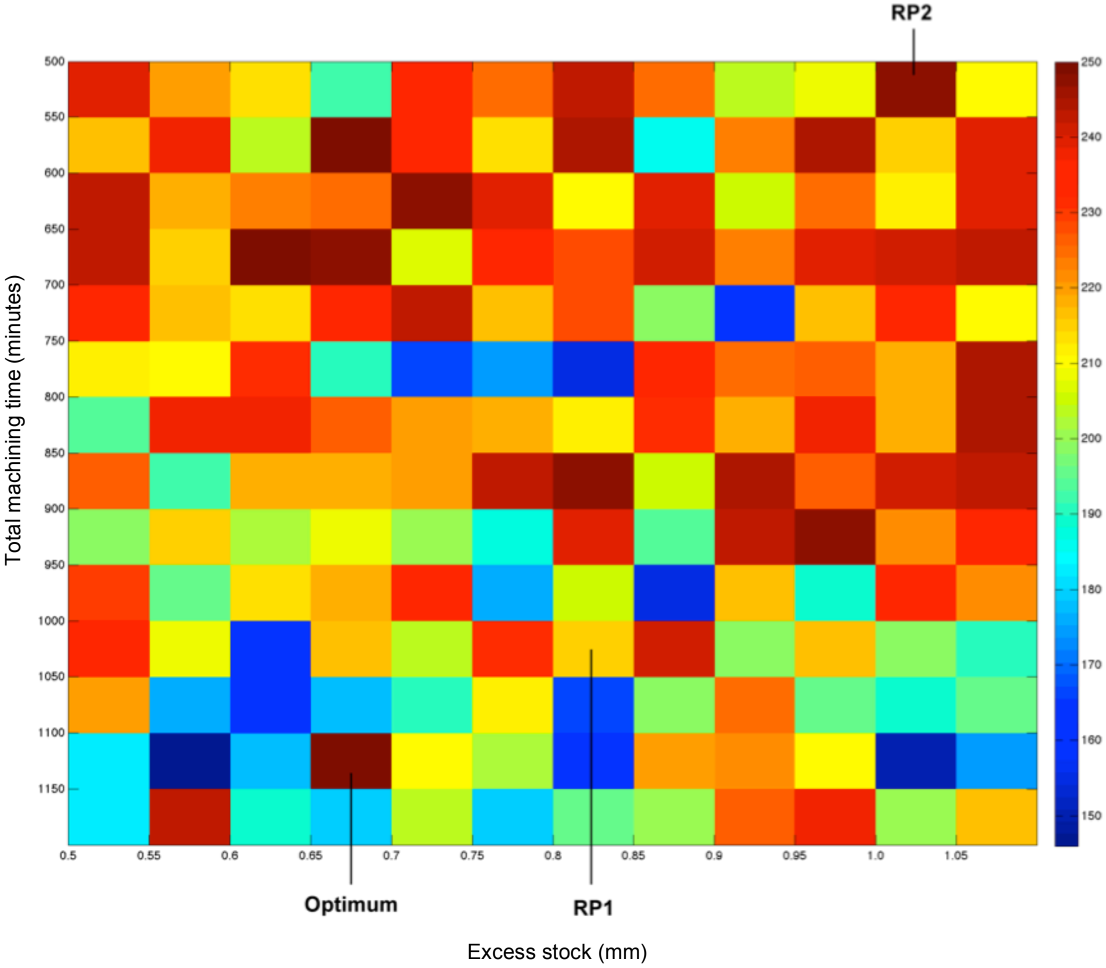
7.4.4 Reference Points

On Part 2-4-H, the difference in performance between the two reference points, RP1 (1,000 mins, 0.8mm) and RP2 (500 mins, 1.0mm) was a surprising result. This is because RP1 is closer to the optimal solution of 1,152 minutes and 0.6mm but achieved worse results. To investigate this further, 168 reference point configurations were tested over 250 runs each on Part 2-4-H. We test different values for total machining time, starting from 500 minutes and increasing in increments of 50 minutes until 1,150 minutes. For each total machining time tested, we also test different excess stock values, starting at 0.5mm and going up to 1.05mm. The configurations used a population size of 15 for 500 evaluations. The results are seen in Figure 7.14. Again, RP2 performs significantly better than RP1.

The pseudo-colour plot shows that there is a great deal of variation in search performance among different reference point configurations. To a certain degree this is to be expected. In the multi-objectivisation case, there is a single best solution. Different reference points will guide search to different regions of the Pareto front. If the best solution is not in that region then it is unlikely that it will be discovered. However, Figure 7.14 shows that on Part 2-4-H, similar reference points can have very different success rates. For example the reference point, 1100 minutes and 0.65mm excess stock, is the closest tested point to the optimal solution. This reference point has a near perfect success rate but its immediate neighbours are very poor in comparison. This suggests that on this search space, with this evaluations budget, a reference point is very sensitive to change.

There could be many reasons for this. Firstly, as can be seen in Figure 7.12, different reference points can cause the search process to take longer to converge, needing to use more evaluations. This could explain why some populations perform so poorly. A similar reference point may also direct search to a suboptimal Pareto front, and with the large number of local optima presented in the deceptive search space of Part 2-4-H, it may be difficult to find routes to the optimal solution from these positions. Solutions that are attractive earlier in the search process may have different properties to the optimal solution and prevent it from being easily reached. Also, locally optimal points close to the reference point may cause search to converge

Figure 7.14. Pseudo-colour plot showing the number of optimum solutions found on the hybrid Part 2-4-H over 250 runs of R-NSGA-II using 168 reference points defined with total machining time in minutes on the y-axis and excess stock in mm on the x-axis. The total machining time in minutes starts at 500 minutes and increases in increments of 50 minutes. Excess stock starts at 0.5mm and increases in increments of 0.05 mm. The colour bar on the right shows the number of optimum solutions found, with dark red being the most and dark blue being the least.

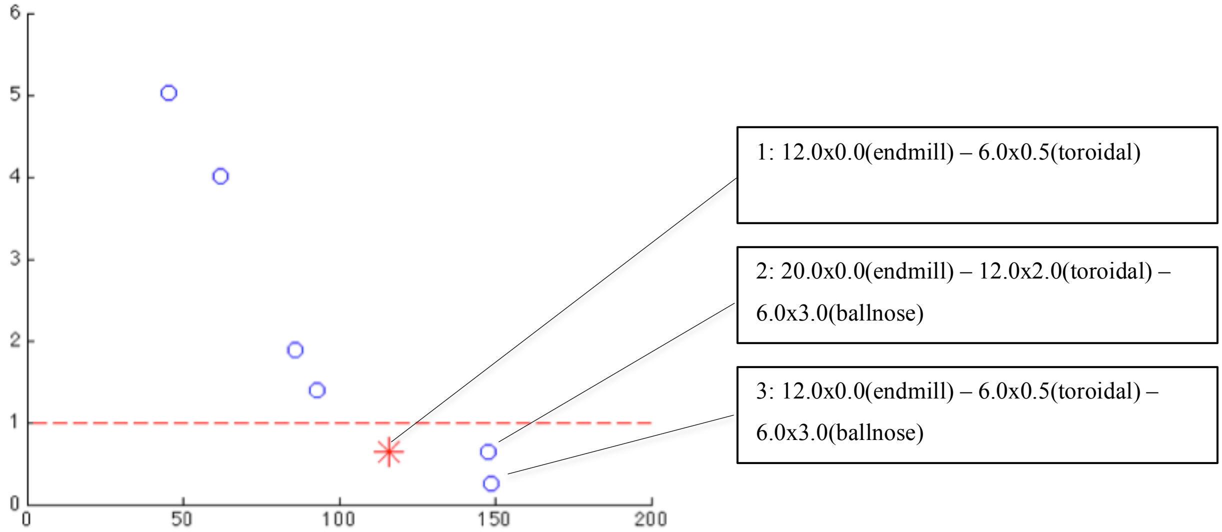


prematurely. This could suggest that a more abstract, looser guidance method such as Guided Elitism could be better, as it maintains a large exploratory range but still leads search to interesting areas, without the sensitivity problem that different reference points seem to have on Part 2-4-H.

7.4.5 Finding Multiple Solutions

The results discussed above show that multi-objective algorithms are able to perform better than single-objective algorithms on a Tool Selection problem with a difficult search space. Another advantage of these algorithms is their ability to locate multiple solutions in a single search run. We will now discuss the Pareto fronts obtained by the algorithms on the two different parts analysed in this chapter.

Figure 7.15. Showing the components of the tool sequences that compose three interesting points on the Pareto front for Part 1.



7.4.5.1 Part 1

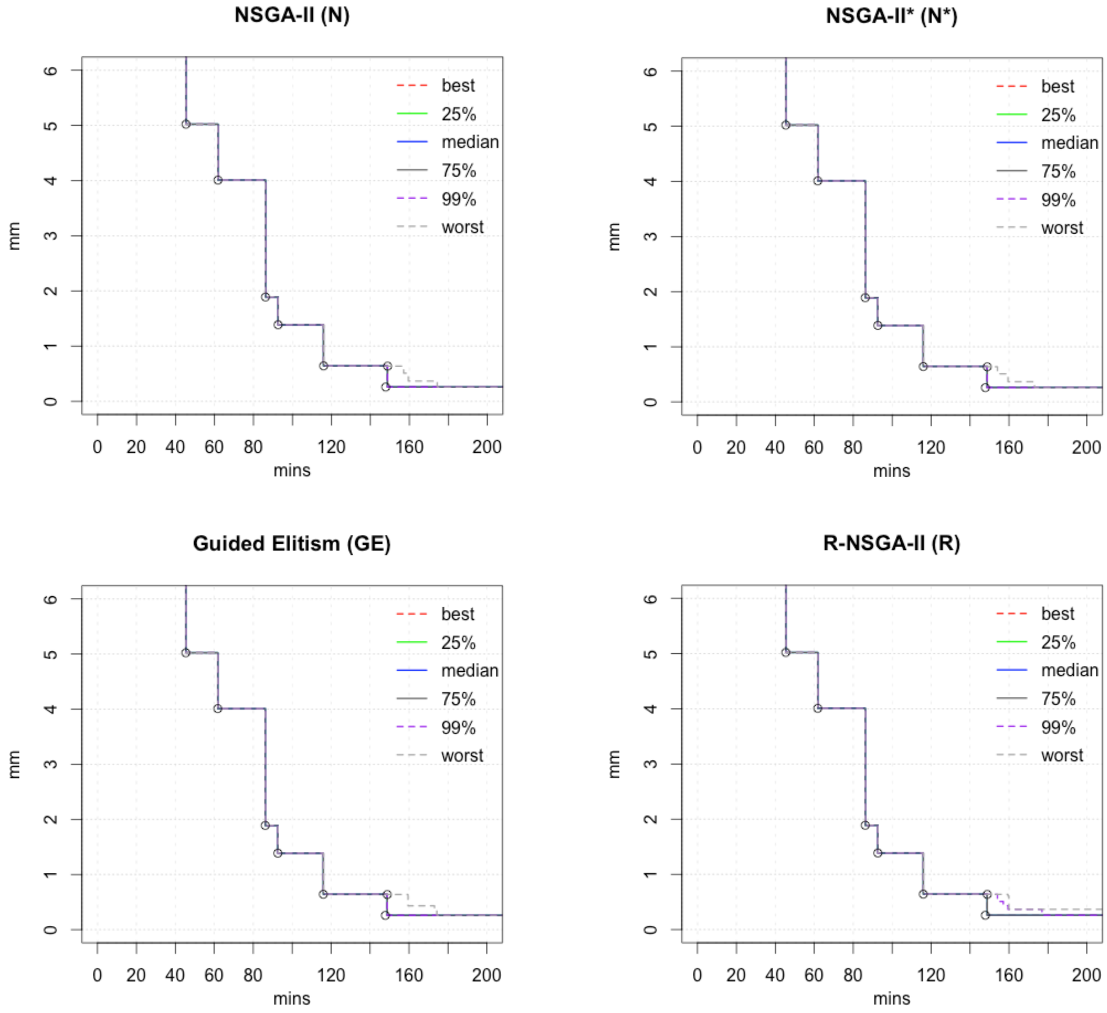
In Figure 7.15, we have highlighted the three Pareto optimal points that achieve a maximum excess stock under 1mm. Solution 1 is the single-objective optimal solution, consisting of a 12mm end mill followed by a 6mm toroidal cutter. Solution 2 uses completely different tools to Solution 1, employing a 20mm end mill, a 12mm toroidal and a 6mm ball nose cutter. This could offer an interesting alternative to Solution 1, as it spreads the load onto larger, stronger tools and in doing so reduces the risk of tool breakages. However, this takes 35 minutes longer and so may be undesirable. Solution 3 uses the same sequence as Solution 1 but with an extra tool at the end, a 6mm ball nose. This solution informs the decision maker (DM) that using this sequence will considerably reduce the maximum excess stock but as with solution 1, at the cost of 35 extra minutes of machining time. This could influence the DM’s choice as it could affect which tool is used in the finishing process.

In Figure 7.16, we see Empirical Attainment Function (EAF) plots for the four algorithms, applied to Part 1 with 200 evaluations and a population size of 15. These show the surface captured by the best, top 25%, median, bottom 25% bottom 1% and worst surfaces attained over the 1,000 runs. From these plots we can see that in over 99% of cases, N, N* and GE are able to achieve all three of these solutions in a single run. R also regularly finds these points but in the bottom 1% of cases finds dominated solutions close to 2 and 3.

7.4.5.2 Part 2-4-H

In Figure 7.17 four interesting tool sequences are highlighted on the Pareto front for Part 2-4-H. Having a selection of solutions is useful for the decision maker (DM). These results give an understanding of the search space. For example, it suggests that the 12 and 16mm end mill tools are important for doing the early bulk of the tool material removal. The 12mm toroidal cutter is also an important tool, appearing in

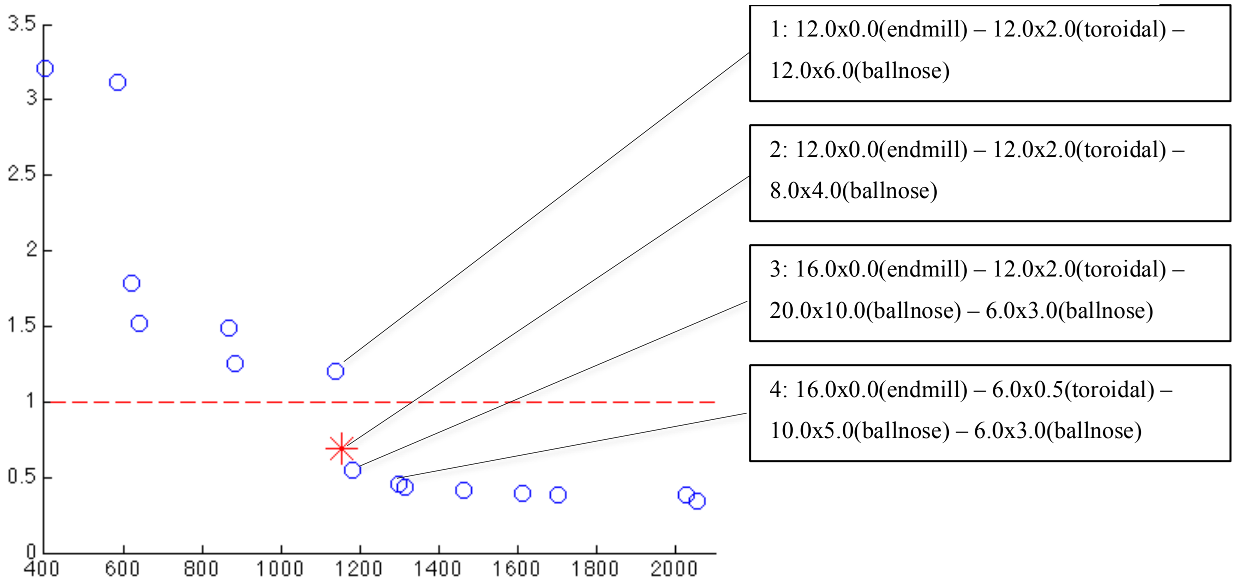
Figure 7.16. Empircal Attainment Function plots for the multi-objective algorithms on Part 1. Circles show the positions of Pareto optimal solutions.



three out of the four highlighted sequences. It also suggests that a ball nose tool is needed to reach the desired surface tolerance.

As we have seen with Part 1, if the DM has access to several Pareto optimal solutions, a more informed choice can be made. For example, the solution that we have considered optimal from the single-objective perspective takes around 1,150 minutes with 0.7mm of excess stock. The next fastest solution achieving the desired tolerance takes around 1,180 minutes to machine with 0.5mm of excess stock. Looking at the tool sequences that compose these solutions, the slower solution may be more attractive because it has lower excess stock. It also finishes with a smaller tool, which is generally cheaper. There may also be extra factors that are not included in the model. For example, the DM could know that a 12mm end mill tool is needed for the next job, and would prefer a solution that avoids this tool on the first job, in the presence of an acceptable alternative.

Figure 7.17. Showing the components of the tool sequences that compose four interesting points on the Pareto front for the hybrid Part 2-4-H.



1: 12.0x0.0(endmill) – 12.0x2.0(toroidal) – 12.0x6.0(ballnose)

2: 12.0x0.0(endmill) – 12.0x2.0(toroidal) – 8.0x4.0(ballnose)

3: 16.0x0.0(endmill) – 12.0x2.0(toroidal) – 20.0x10.0(ballnose) – 6.0x3.0(ballnose)

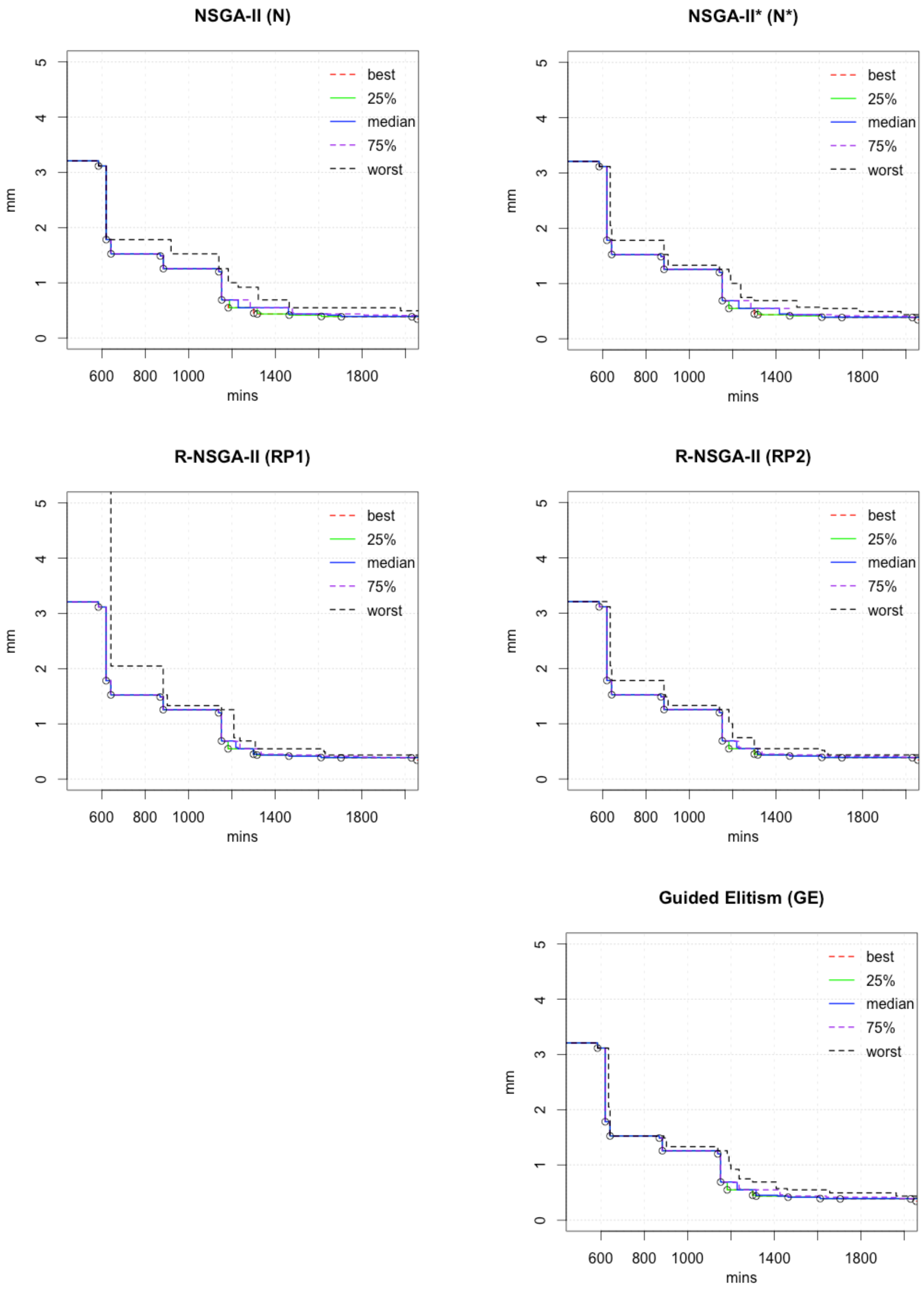
4: 16.0x0.0(endmill) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose) – 6.0x3.0(ballnose)

Figure 7.18 and Figure 7.19 below show EAF plots for the multi-objective algorithms on Part 2-4-H for each algorithm using a 500 evaluations limit and a population of 15. These plots give a good indication of the solutions returned at the end of multi-objective search. Figure 7.19 shows that even in the worst case, all of the algorithms return a reasonable approximation of the Pareto front, with many solutions presented to the DM. Their performance is also similar, especially looking at the median attained surfaces.

Figure 7.19 shows the attained surfaces in the region below 1mm of excess stock. It is clear that the methods with preferential search obtain slightly better performance. Referring to Figure 7.19, we can see that the preferential methods attain surfaces much closer to the Pareto front than N and N*. However, it is interesting that in the median case, while all of the algorithms return Solution 2, our single-objective optimum, none of the algorithms are able to return Solution 3. Both reference point methods and GE are able to achieve both solutions in the top 25% of runs but N cannot. It seems that the algorithms frequently converge to a suboptimal Pareto point, dominated by Solution 3. The reference point methods are able to achieve Solution 4 more frequently than GE.

While the tested algorithms perform well and on average are able to find a good approximation of the true Pareto front, there may be ways that this could be improved. Preferential search appears to have improved convergence on desired areas. Including more reference points, or using a co-evolutionary guidance method, such as in (Wang et al., 2013), may allow faster or more consistent convergence to the Pareto front. However, the algorithms achieve a good approximation given the small evaluations budget, and the knowledge that there are several local optima on this part. To capture the entire front they may need to use larger population sizes. In Figure 7.12 we see that all of the algorithms deteriorate in performance with larger population sizes, which suggests that they need to be run for longer in order to fully converge in this case.

Figure 7.18. Empirical Attainment Function plots for the multi-objective algorithms on the hybrid Part 2-4-H. Circles show the positions of Pareto optimal solutions.



NSGA-II (N)

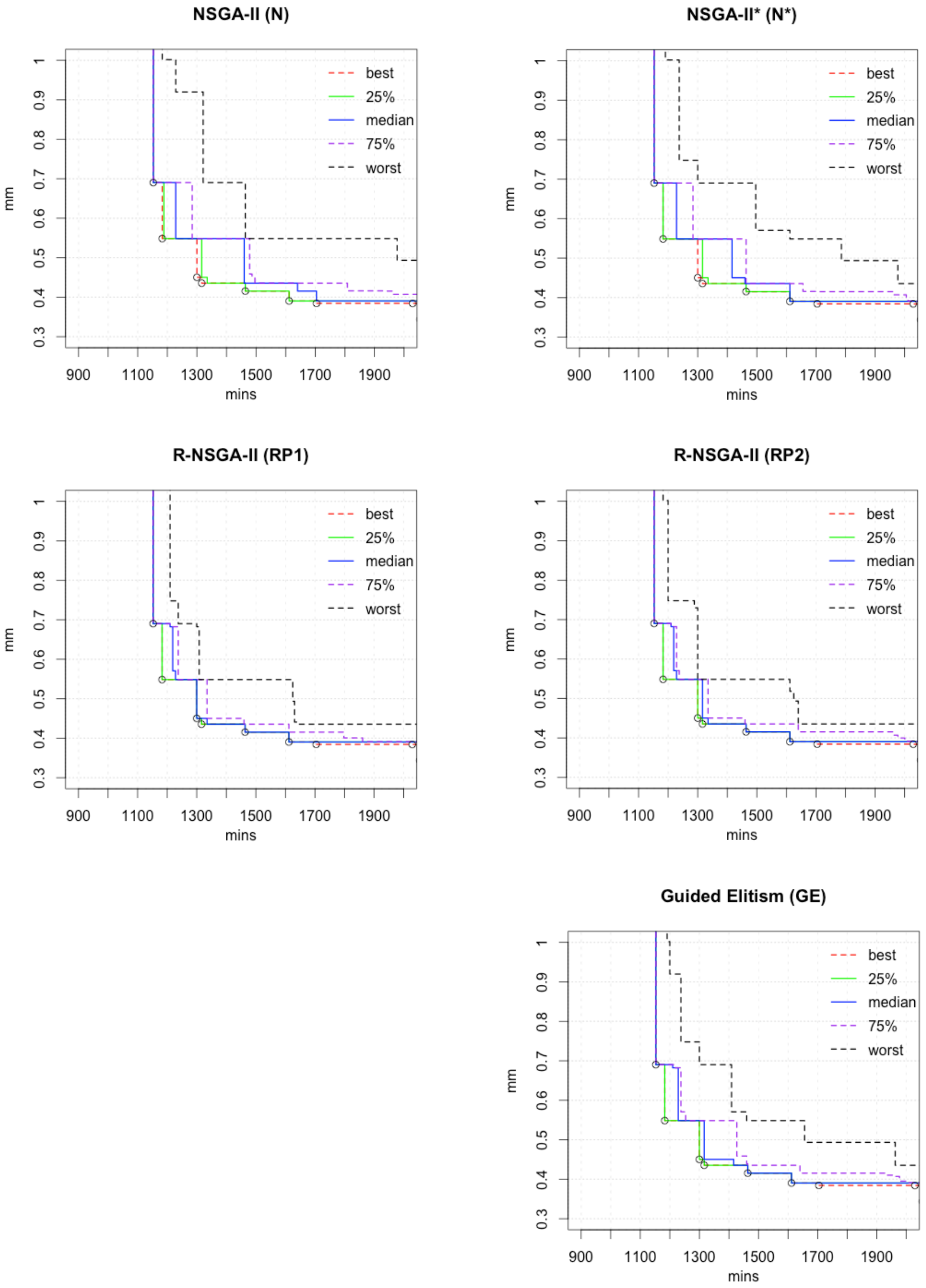
NSGA-II* (N*)

R-NSGA-II (RP1)

R-NSGA-II (RP2)

Guided Elitism (GE)

Figure 7.19. Empirical Attainment Function plots for the multi-objective algorithms on Part 2-4-H, zoomed in at solutions with excess stock under 1mm. Circles show the positions of Pareto optimal solutions.



7.5 Discussion

The results show that ‘multi-objectivised’ search algorithms perform well on both a moderate and difficult search space, given a strict budget of evaluations. On the moderate search space of Part 1, the GA performs equally well and with less variation between different configurations. This suggests that multi-objective search is not needed on this component, when the focus is on finding a single solution. However, more information is offered in the multi-objective case, and we see that on this search space the multi-objective algorithms are able to find all of the Pareto optimal solutions with extremely high frequency.

On the difficult search space of Part 2-4-H, multi-objective search vastly outperforms both single-objective algorithms. This is particularly the case for higher evaluation limits and success is still rather small at the 150 evaluations level, with the best algorithms finding the optimum solution around 30% of the time. With a limit of 350 evaluations, the best results showed a success rate above 90% and at 500 evaluations all of the multi-objective algorithms have configurations that can find the optimum solution at least 99% of the time, with their best configurations.

NSGA-II performed competitively on Part 2-4-H but the modifications introduced in this chapter, Guided Elitism and the sorting correction achieved an even better success rate. The colour plots in Figure 7.11 show that for this problem, smaller population sizes are more effective when using tight evaluation limits. GE performed very well on Part 2-4-H and most configurations were considerably more successful than NSGA-II. This suggests that for this difficult search space, GE offers a good method for providing search preferences. Interestingly, on Part 1 NSGA-II is very successful for most configurations, and appears to converge quicker than GE. This could be due to the smaller size of the front present in the Part 1’s search space, and implies that non-biased search works better there. GE seems to perform badly with very small population sizes, which could be due to a lack of diversity encouraged by the elitism.

Although the exact solutions are not industrially relevant, the hybrid Part 2-4-H offers an interesting search space to test these algorithms. One advantage is that it provides a larger Pareto front than Part 1. Using this, interestingly we see that the multi-objective algorithms are able to frequently get very close to the Pareto front, even in the presence of many local optima. Unlike with Part 1, the algorithms using preferential search also provided a better selection of solutions in the desired region. Using this search space with many local optima, we also saw a large difference in success rate between the two reference points in RP1 and RP2. The worse method uses a reference point that is very close to the optimum but the algorithm seems to converge on a suboptimal solution. However, the other reference point, RP2 is arguably the most successful search algorithm, so the reference point method is clearly an important tool to use on this type of problem. Comparing many reference points shows that there is a lot of variability on this search space, which potentially could be avoided by using multiple reference points or adapting the points towards the end of search.

On both components we see that multi-objective search needs more evaluations to converge. This could mean that with a very small evaluation budget they are impractical. However, given a reasonable and still low number of evaluations they offer the advantages of much better performance on a difficult landscape and extra information to the decision maker.

7.6 Summary

In this chapter, multi-objective techniques were applied to single-objective tool sequence optimisation. Comparing single and multi-objective approaches, the multi-objective algorithms were shown to perform considerably better on the difficult multi-modal test search space present in Part 2-4-H. On the simpler search space present in Part 1, the performance of the two approaches was evenly matched, although the single-objective GA performs better with a very low evaluation limit and is less sensitive to population size changes.

Including preferences in search improved performance, and this was especially noticeable on the difficult search space. From a single-objective perspective, the optimal solution was found most often by the R-NSGA-II and Guided Elitism algorithms. The choice of reference point affected the frequency that the single-objective optimal solution was found. However, from a multi-objective perspective R-NSGA-II provided the best overall approximation of the Pareto front, closely followed by Guided Elitism. Presenting the decision maker with multiple solutions could prove very useful, as a more informed decision can be made. This shows two advantages to the multi-objective approach – better performance on a difficult search space and the return of several interesting solutions. In the next chapter, this work is extended to the multi-objective case, by including an extra objective, total tooling costs. Here, we will be explicitly evaluating algorithms on the range of solutions they can return in a single search run.

Chapter 8

Multi-objective Tool Sequence and Machining Parameter Optimisation

In the previous chapter, multi-objective methods were applied to the Tool Selection Problem, but it was still largely approached from a single-objective perspective. The multi-objective formulation enabled search to better avoid getting stuck in local optima, while also providing a selection of solutions that could provide useful alternatives to decision makers. While machining time and surface tolerance are two important objectives in machining operations, there are many other considerations that could affect the manufacturing process. In this chapter an extra objective, total tooling costs, is included in the optimisation problem and an important machining parameter, cutting speed, is included in the decision variables. This represents the first time that a multi-objective approach has been taken to tool sequence optimisation and one of very few pieces of work that combines Machining Parameter Optimisation with Tool Sequence Optimisation. The results show that different tool sequences can be used to locate solutions in different regions of the Pareto front, and that a number of algorithms are able to find good trade-offs between total machining time and total tooling costs, which can provide valuable information to machinists. A novel constraint handling method, referred to as Precedential Objective Ordering is shown to produce consistently good results. Parts of this chapter appear in (Churchill et al., 2013b) and (Churchill et al., 2013c).

8.1 Introduction

The aim of the work in this thesis is to provide a system that can automatically provide decision makers with industrially relevant tool sequences. One large assumption has been made in the machining optimisation work presented in previous chapters, that tools will survive for the length of the tool sequence. This assumption is shared by a number of other researchers, such as (Lim et al., 2001; Vosniakos and Krimpenis, 2002; Spanoudakis et al., 2008; Chen and Fu, 2011). It may not be unreasonable, and does not necessarily invalidate the results of the approach. For example, a hard tool may be cutting a soft material such as aluminium, wood or plastic. In this case tools will last for a very long time. Tools may also be coated in elements such as titanium nitride, which can extend tool life by 800% (Astakhov and Davim, 2011). Depending on the job, this can ensure that tools will be able to last for the time needed. Coating with thick-film diamond may double or triple this extension in tool life. Although these tools can be expensive, they are very durable and their working life may be difficult to accurately predict. In this situation, it may be problematic to include tool wear in the search space and as such, optimisation must be made without it (Astakhov and Davim, 2011).

In many circumstances, however, tool life is an important aspect to consider in tool selection problems. Tools cost money and are a finite resource. When they wear out beyond an acceptable limit, they must be replaced. This means that different tool sequences have different tool costs associated with them. Cutting speed, the speed at which the tool moves against the material of the workpiece, is closely tied to tool life (Groover, 2007). Clearly, the total machining time is greatly affected by cutting speed. This means that there is a trade-off between total time and total cost. If we consider a single tool sequence, increasing the cutting speed will decrease the overall machining time and bring the tools closer to retirement or break them.

The trade-off between time and cost is a classic multi-objective decision problem. In this chapter cutting speed will be included in tool sequences and a truly multi-objective approach will be taken to solving the tool selection problem. The goal will be to return a diverse selection of Pareto optimal solutions, each of which representing a good trade-off between time and cost. Two groups of experiments are performed. The first uses real world tool life data on a simple library of solely end mill tools. The second uses estimated tool life data on a more complex library containing end mill, toroidal and ball nose tools.

8.2 Background

In this section we will first discuss tool life and how it is measured before describing how we will integrate it into our system as a new objective. We will then review related work that has been carried out in this area.

8.2.1 Tool Life

Machining operations offer a harsh environment for cutting tools. They are subject to a great deal of force, while eroding a surface at fast speeds leads to very high temperatures. If either of these conditions is too great they can cause catastrophic failure, where the tool breaks. This can damage the work surface, and so is avoided at all costs by working within safe parameter constraints.

Given the nature of their job, tools unavoidably wear out over time. This is known as *gradual wear* (Groover, 2007; Kalpakjian and Schmid, 2008). There are many processes that can contribute to the deterioration of the cutting edge of the tool, which reduces the accuracy and surface quality that the tool can produce. These include abrasion, where the hard surface of the workpiece breaks off small amounts of material from the tool; adhesion, where high temperatures cause the parts of the tool to weld together with the workpiece; and plastic deformation, where forces acting at high temperatures cause the shape of the cutting edge to change (Groover, 2007; Black and Kohser, 2012).

Tool life is the amount of cutting time that a tool can be usefully employed. It is a measure of the duration that the tool can be used for before the deterioration of the cutting edge becomes unacceptable. Both depth of cut and feed contribute to tool life but by far the most important parameter affecting it is the cutting speed (Groover, 2007; Kalpakjian and Schmid, 2008; Black and Kohser, 2012). *Taylor's tool life equation* describes the relationship between a tool's useful cutting time in minutes, T, and the cutting speed in mm/s, V_c, as:

$$V_c T^n = C \qquad (8.1)$$

where n and C are constants found experimentally, and related to the material of the tool and workpiece. C defines the speed where T = 1 minute (Kalpakjian and Schmid, 2008).

8.2.2 Optimising Tool Sequences with Tool Life Consideration

In previous chapters, tool selection has been framed as a discrete combinatorial problem. Each tool was assigned fixed machining parameter values based on manufacturer's recommendations. However, Taylor's Tool Life Equation (equation 8.1) shows that different cutting speeds can shorten or extend tool life, as well as affecting total machining time. In the experiments in this chapter, we will simultaneously optimise tool sequences and the cutting speeds for each tool in a sequence. This adds continuous properties to each sequence. Rather than a sequence having one finite value for machining time, it can now have a range of different possible machining times, dependent on different cutting speeds. The faster the cutting speed, the faster the tool moves along the tool path and the shorter the machining time.

The search space now consists of the discrete tool sequences, which define the length of the tool paths, and continuous cutting speeds, which define how fast a tool moves along the path. With no consideration of tool wear, cutting speed could be set to the maximum allowed values and the problem could be reduced back to the discrete one. However, in the real world tools have a limited lifetime and a monetary cost. There is also a time cost associated with changing a tool. This means that using a high cutting speed that consumes just over one tool for a particular tool path can take longer than a low speed that consumes less than one tool.

In the experiments below, Tool Sequence Optimisation will be framed as a multi-objective problem. The algorithms need to find solutions with good trade-offs between minimising two objectives – total machining time and total tooling costs. The tested algorithms will also be required to return solutions that meet a desired value for maximum excess stock.

8.2.3 Related Work

In the literature, which was reviewed in Chapter 2, there has been a distinction between research looking at tool selection and research focused on cutting parameter optimisation. Only three systems have been found in the literature that combine the two tasks. In (Spanoudakis et al., 2008), the depth of cut (the distance that the tool penetrates the workpiece) is optimised at the same time as selecting a tool, on a milling operation. Tool sequences are restricted to being exactly two tools in length and there is no consideration of tool wear. In (Wang and Jawahir, 2005) a turning operation is optimised. Cutting speed, depth of cut, feed and tool diameters are all included in the decision variables. Weightings are used in the objective function, and the user can determine whether they prefer maximising tool life or minimising surface roughness. Again, only fixed length two-tool sequences are considered in the optimisation. In (Krimpenis and Vosniakos, 2008), a multi-objective approach is taken to tool selection with cutting parameter optimisation, finding a Pareto front formed of three objectives – total machining time, surface tolerance, and surface uniformity. The goal of the search is to find the set of single tools, with associated cutting parameters, that are Pareto optimal with respect to the three objectives. Only single tools are selected, so the problem of determining optimal multiple-tool sequences is ignored. Tool wear is considered as a constraint but tooling costs are not included as an objective.

In the experiments below we introduce the first system that offers multi-objective, multiple-tool optimisation on a machining task, with the inclusion of an important machining parameter. In a similar vein to previous chapters, it is explored on rough end milling of a sculptured surface. Compared to the papers described above, much more freedom is allowed in terms of tool sequences. The multi-objective approach allows many different trade-off solutions to be returned to the user in a single run, while in (Wang and Jawahir, 2005) and (Spanoudakis et al., 2008), preferences need to be set through a priori weightings. Tooling costs are included as an objective, which is not considered in (Krimpenis and Vosniakos, 2008) and (Spanoudakis et al., 2008).

8.3 Methods

In this section we will first describe the way in which tool sequences are evaluated in the experiments below. We then present a new representation scheme and search operators that have been modified from those found in the previous chapters to work in the discrete and continuous space of the problem tackled in this chapter. In the last part of this section we describe the algorithms tested in the experiments below.

8.3.1 Evaluation of Tool Sequences

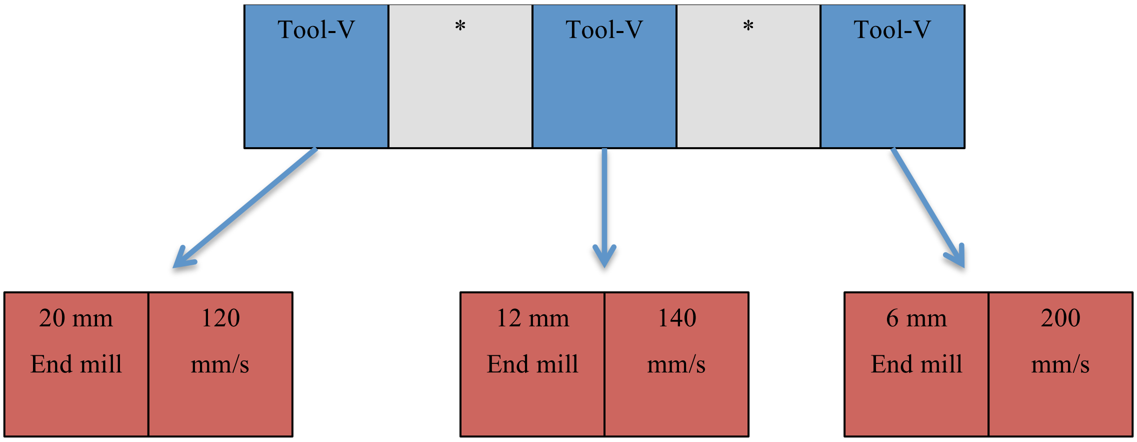
The goal of the search in these experiments is to find tool sequences with a good trade-off between total machining time and total tooling costs. These sequences also have to operate under the constraint of having a maximum excess stock below a specified target. In the experiments below, this is going to be treated as a constraint that is included as an objective. To this end tool sequences are evaluated with three independent objectives – total machining time, total tooling costs, and maximum excess of stock. However, we only want to return solutions to users within the desired excess stock limit that are on the bi-objective Pareto front formed by total machining time and total tooling costs.

The total machining time is calculated by taking the tool paths (distance) for each tool and dividing that by the appropriate cutting speed. The tool path is divided into four cuts or movements. The cutting movement carries out the main material removal. When the tool moves between positions and is not in contact with any material, it uses rapid movements. Lead-in and lead-out movements are used to enter and exit the part without adversely affecting the surface. The total machining time, T_m, for a sequence, $\boldsymbol{E}$, on a part p, is calculated using $f_m(\boldsymbol{E}, p)$ described below:

$$f_m(\boldsymbol{E}, p) \; = \; \sum_t^n (C_t \, . \, f_t) + \; (Li_t \, . \, 0.5 f_t) + \; (Lo_t \, . \, 0.5 f_t) + \; (R_t \, . \, 10{,}000) + T_{ch} \quad (8.2)$$

where for each tool, t, C_t is the cutting tool path, Li_t is the lead-in tool path, Lo_t is the lead-out tool path, R_t is the rapid tool path and f_t is the tool's table feed. The tool paths are in mm and the table feed is in mm/min. If there is a tool breakage, then an assumption is made that the lead-in of the tool needs to be repeated to enter the part. This increases the total machining time and also creates a more complicated interaction in the search space, as large jumps can be created by small changes in cutting velocity.

Figure 8.1. Showing an example of the new representation scheme for tool sequences with associated cutting velocities. A three-tool sequence is shown with associated machining parameters.



Generated Tool Sequence Plan:

1. 20mm End mill @ 120 mm/s
2. 12 mm End mill @ 140 mm/s
3. 6 mm End mill @ 200 mm/s

Total tooling costs are calculated by dividing the total machining time for each tool by the estimated tool life based on *Taylor's tool life equation* described above in equation 8.1. This gives the number of tools used, which is multiplied by the cost for the tool. The total tooling cost, T$_c$, is given by the equation,

$$T_c = \sum_t^{t_n} \left[\left(\frac{C'_t}{Tl_{vt}} \right) + \left(\frac{Li'_t}{Tl_{0.5vt}} \right) + \left(\frac{Lo'_t}{Tl_{0.5vt}} \right) \right] . T_{cost} \qquad (8.3)$$

where, for each tool, t, C'_t is the time taken by the cutting operation, Li'_t the lead-in operation, Lo'_t the lead-out operation, and Tl_{vt} is the tool life at V$_t$. If more than 100% of a tool's lifetime is required, an extra lead-in and tool change time is added for each additional whole tool needed.

The third objective is the maximum distance of excess stock, which is provided by the CAM software, *Machining Strategist* (Vero Software, 2012). The tool paths and stock models used by the evaluation functions are produced by the CAM software, which is an expensive operation. Tooling costs however are calculated using Taylor's tool life equation. This is a much cheaper calculation and as such a greater number of evaluations are allowed in the experiments below.

8.3.2 Representation

The representation scheme used on Tool Sequence Optimisation in previous chapters was based around a list of L elements, where L is the maximum length of tool sequences. Each element contained either a tool or an ignore symbol (*). The sequence is read from left to right, and if an ignore symbol is encountered it is ignored. In this way, variable length tool sequences can be generated between one and L tools.

To add cutting speed (*V*) information into the sequence, the above representation has been modified. Each tool is now bonded with an extra element, containing *V* as a floating-point number in mm per minute. Each tool and *V* pair is considered to be a chunk that cannot be separated from each other. The representation still uses L elements but each non-junk element is a pointer to a sub list containing the tool-V pair. This can be seen in Figure 8.1. In the experiments in this chapter, the tool sequences are limited to five tools. This is not a limitation of the representation scheme, which could accommodate any tool sequences of any length. This restriction is in place because a cached version of the search space is used, and the limit of five was chosen as a good balance between realistic, useful tool sequences and the time taken to simulate the entire search space.

8.3.3 Recombination

To accomplish recombination, one chunk (tool-cutting speed pair) can be copied from one parent to another. The crossover procedure works in exactly the same way as the method found in Chapter 5. The only difference is that instead of moving only a tool (or ignore symbol), now a tool with its associated cutting speed is copied.

8.3.4 Mutation

In the experiments below, mutation is carried out in two different ways. The gradual mutation method described previously in Chapter 5 is used to mutate the discrete part of the tool sequence. This means that a tool in the sequence can be changed or removed, or a new tool is inserted. In the case of an insertion or a tool change, a new chunk is created containing the new tool and a default value (chosen using handbook suggestions) for the cutting velocity with a uniform random value of between ±5 m/min added to this.

For the continuous part of the tool sequence, the cutting speeds of the individual tools, there is a $\frac{1}{tool\ sequence\ length}$ chance of a random value from a Gaussian distribution, $\mathcal{N}(0,1)$, being made to the cutting speed of each tool in the sequence. This occurs regardless of whether a discrete mutation has been made. Each tool library contains information on the minimum and maximum cutting speeds that can be used based on manufacturer recommendations. If a mutation produces a value outside of these limits, the value is capped at the limit.

8.3.5 Algorithms

Five multi-objective algorithms are tested on this problem. The first of these is an unconstrained version of NSGA-II (Deb et al., 2002), and the other four are variants of this algorithm. The modified versions have adapted NSGA-II in order to guide it towards solutions that satisfy the excess stock constraint. The preferential search strategies that were tested on the multi-objectivisation problem in the previous chapter, R-NSGA-II (Sundar and Deb, 2006) and Guided Elitism, are now applied to this multi-objective problem. Additionally, two methods are tested for constraint handling, that both modify the dominance relationship. The first is weighting the objective values (*Weighted Objectives)* and the second is a novel modification to NSGA-II that will be referred to as *Precedential Objective Ordering*.

Table 8.1. Two example solutions showing untouched values for the three objective functions.

Solution	$\boldsymbol{f_1(S)}$	$\boldsymbol{f_2(S)}$	$\boldsymbol{f_3(S)}$
1	1000	2000	1.3
2	1500	3000	1.1

Table 8.2. Two example solutions with the third objective function, $\boldsymbol{f_1(S)}$ modified to have a goal at 1.3.

Solution	$\boldsymbol{f_1(S)}$	$\boldsymbol{f_2(S)}$	$\boldsymbol{f_3(S)}$
1	1000	2000	1.3
2	1500	3000	1.3

8.3.5.1 NSGA-II

The NSGA-II algorithm has been previously described in Chapter 6, and used in experiments in Chapter 7. It is used here without explicitly handling constraints. There are three objectives:

$$f_1(\boldsymbol{S}) = \text{minimise total machining time} \qquad (8.4)$$

$$f_2(\boldsymbol{S}) = \text{minimise total tooling costs} \qquad (8.5)$$

$$f_3(\boldsymbol{S}) = \text{minimise the maximum distance excess of stock} \qquad (8.6)$$

where $\boldsymbol{S}$ is a vector of the decision variables, which are the tools and their respective cutting speeds. The unconstrained algorithm does not punish non-dominated solutions that are above our required threshold for $f_3(\boldsymbol{S})$. However, solutions that minimise that objective value will be non-dominated, so the algorithm should be able to return interesting and relevant solutions as part of finding a good approximation of the Pareto front.

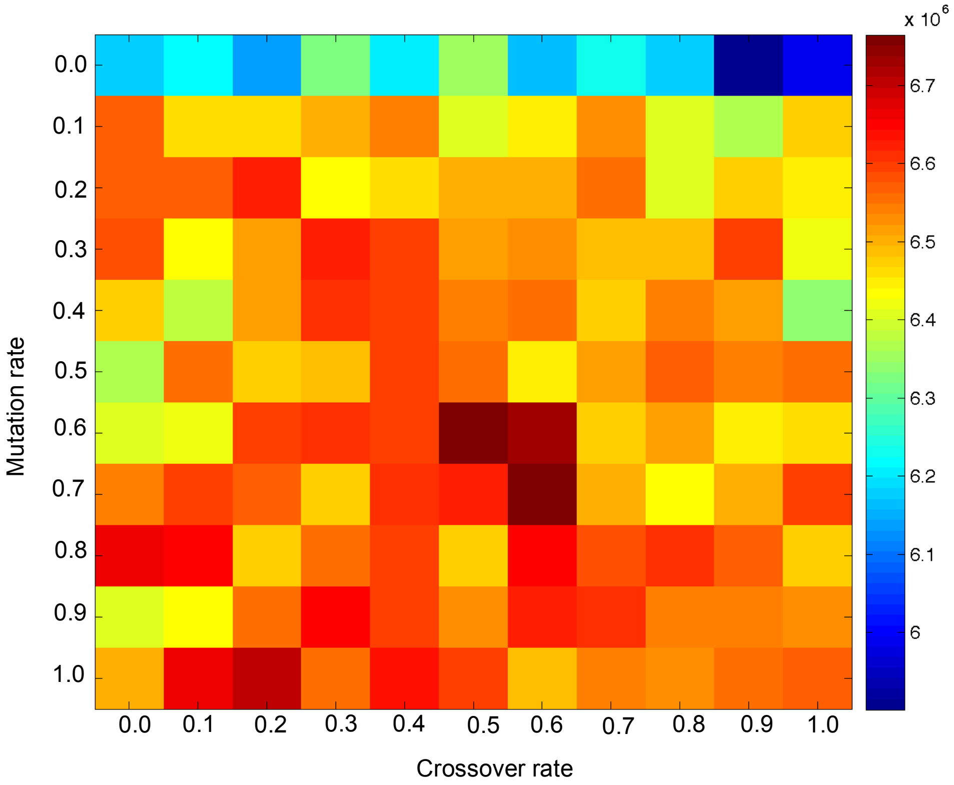
In the experiments in this chapter, a target value is provided to the algorithm for $f_3(\boldsymbol{S})$. In contrast to Chapter 7, in these experiments we do not want to differentiate between solutions under the required surface tolerance. To prevent solutions from being treated favourably based on differences in excess stock under the target value, once the target has been reached this objective value is fixed at the limit. For example, let us consider the two solutions in Table 8.1.

Neither of the solutions dominates the other. However, if the goal is to have solutions with a value for $f_3(\boldsymbol{S}) \leq 1.3$, Solution 1 clearly dominates Solution 2. Using the target scheme describe above, values for $f_3(\boldsymbol{S})$ under 1.3 are fixed at this value. The modified solutions are shown in Table 8.2, and we can see that Solution 1 now dominates Solution 2. Where $max(\boldsymbol{d})$ is the maximum distance of excess stock, and the target tolerance is k, adding this limit to $f_3(\boldsymbol{S})$ gives,

$$f_3(\boldsymbol{S}) = \max(\max(d), k)) \qquad (8.7)$$

The other algorithms tested in the experiments below make small modifications to this version of NSGA-II. A parameter sweep was carried out on this base version to find the best parameters to use in the experiments below. Configurations were tested for 100 runs on Part 4, with an 18-tool library, in the same implementation as described in Section 8.4.2 below. Mutation and crossover rates were tested

Figure 7.2. Results for the parameter sweep for NSGA-II with a population size of 24, on Part 4 with an 18-tool library. The colours indicate the median hypervolume obtained over 100 trials with a 500 evaluations budget.



with values of 0.0 – 1.0 in increments of 0.1. These tests used a population of 24, which was found to work well in preliminary testing. A mutation rate of 0.7 and a crossover rate of 0.6 were chosen for the experiments below. All of the other NSGA-II variants used these parameter values.

8.3.5.2 Weighted Objective NSGA-II

The normal NSGA-II is modified to have weighted objective values, to handle the constraint of keeping the maximum thickness of excess stock below a predefined threshold. This can be considered to be a punishment value, which is added to each of the objectives. This technique is a common constraint handling method and for example has been described in a multi-objective context in (Coello Coello et al., 2004; Watanabe and Sakakibara, 2005; Lochtefeld and Ciarallo, 2012). As we have described in Chapter 7, the benefits of this method are that they discourage infeasible solutions but allow them to be included in the population, which may contain properties that assist in the discovery of good quality solutions later in the search process.

While the penalty function is a relatively simple and inexpensive constraint handling mechanism, it can bias the search in such a way that makes it more difficult to find the Pareto optimal front (Runarsson and Yao, 2005). In order to try to prevent this, the penalty function here takes the form of a weighting on the two unconstrained objectives. Where k is the maximum thickness of excess stock allowed, $f_1(\boldsymbol{S})$ and $f_2(\boldsymbol{S})$ from equations 8.4 and 8.5 are modified using this condition:

If $f_3(\boldsymbol{S}) > k$:

$$f_1(\boldsymbol{S})' = f_1(\boldsymbol{S}) \cdot f_3(\boldsymbol{S}) \qquad (8.8)$$

$$f_2(\boldsymbol{S})' = f_2(\boldsymbol{S}) \cdot f_3(\boldsymbol{S}) \qquad (8.9)$$

where $f_3(\boldsymbol{S})$ is from equation 8.7. This allows for a smooth penalty function, where the penalty value is proportional to the constraint violation. $f_3(\boldsymbol{S})$ is still included as an objective function, to enable solutions that have achieved the desired excess stock to be non-dominated when compared to those that have not reached this target.

8.3.5.3 R- NSGA-II

The R-NSGA-II algorithm has been used previously in chapters 6 and 7. It is chosen for the experiments below because it allows search to be directed towards interesting regions. In this way it provides a way of dealing with constraints without using punishment factors or computationally expensive constraint handling methods.

The version used here is exactly the same as the one described in chapters 6 and 7. In all experiments the reference point (0, 0, 0) is used, corresponding to $f_1(\boldsymbol{S})$, $f_2(\boldsymbol{S})$ and $f_3(\boldsymbol{S})$ from equations 8.4, 8.5 and 8.7 respectively. A weighting of 10 is applied to $f_3(\boldsymbol{S})$, the surface tolerance objective, to encourage feasible solutions to be found. An epsilon value of 0.01 is used to encourage a range of solutions to be returned.

8.3.5.4 Guided Elitism

Guided Elitism has also been described previously in chapters 6 and 7. The experiments in Chapter 7 are based on a multi-objectivisation task, while here the problem is truly multi-objective. It may seem like Guided Elitism is not as useful in this situation but there can be benefits. In certain situations, single objective algorithms may be able to reach the Pareto front more quickly (Ishibuchi and Nojima, 2007). By effectively hybridising single and multi-objective techniques, the search process may find better solutions given strict evaluation budgets. Another benefit of this hybridisation approach is that it allows for the free, unconstrained search provided by NSGA-II, while applying a small amount of guidance towards desired solutions with good trade-offs. This could encourage better convergence on desired results than the unconstrained NSGA-II, while still returning a diverse range of solutions.

The same objective functions are used as in NSGA-II, $f_1(\boldsymbol{S})$, $f_2(\boldsymbol{S})$ and $f_3(\boldsymbol{S})$ from equations 8.4, 8.5 and 8.7 respectively. In addition a small number of solutions are given the highest Pareto rank and a higher crowding distance than non-elite solutions. As in Chapter 7, the number of solutions that are included in the new population based on the single-objective aggregate function is based on this formula:

$$y = round(\frac{population\ size}{10}) \qquad (8.10)$$

For the population size of 24, chosen in the parameter sweep described above, this means that two solutions are automatically added to the new population. The function used to decide which solutions to apply Guided Elitism to is,

$$f_{guided}(\boldsymbol{S}) = f_1(\boldsymbol{S})/\max[f_1(\boldsymbol{S})] + f_2(\boldsymbol{S})/\max[f_2(\boldsymbol{S})] + (f_3(\boldsymbol{S}) \,.\, 10{,}000) \qquad (8.11)$$

$\max[f_1(\boldsymbol{S})]$ and $\max[f_2(\boldsymbol{S})]$ refer to the maximum values for the respective values seen so far in a run. This adds a large punishment to solutions that do not achieve the desired excess stock tolerance, encouraging solutions to reach this target value, with a good trade-off between objective values.

8.3.5.5 Precedential Objective Ordering

This is a novel adaptation to NSGA-II that modifies the dominance rules used for Pareto ranking. It is influenced by other algorithms that modify Pareto dominance rules, such as Preference Order Ranking (di Pierro et al., 2007), G-MOEA (Branke and Deb, 2004), Preferability (Fonseca and Fleming, 1998) and Coello Coello and Toscano's Micro-GA (2001).

The idea behind this approach is that preferences and constraints should be dealt with first, before applying Pareto ranking to solutions. This is similar to the constraint handling technique described in (Deb, 1999) for single-objective optimisation problems, and is used in a multi-objective context in (Coello Coello and Toscano, 2001). In (Deb, 1999), binary tournament selection is used. Choosing a winner between solutions x and y is determined using the following rules in order:

1) If one solution is feasible and the other is infeasible, the feasible solution wins
2) If both are feasible, the solution with the best objective function wins
3) If both are infeasible, the solution with the lowest constraint violation wins

This framework is extended in this algorithm. Instead of simply dealing with constraints, preferences can be dealt with as well. This is accomplished by having a list, L, of constraints and preferences (referred to as conditions), ordered by their importance. A condition can allow solutions to violate constraints by a small amount and to control the sensitivity of preferences. For example, we may prefer one solution to another if it has a machining time of more than 100 minutes but not if this difference is a few seconds.

The dominance assignment between two individuals is described in the pseudo-code in Algorithm 8.1 below. First a lexicographically ordered list, $\mathcal{L}$, of conditions must be specified. "$x.\mathcal{L}_i$" represents the solution x's value for the i^{th} condition of $\mathcal{L}$. When comparing two solutions x and y, x dominates y if $x.\mathcal{L}_i$ satisfies the condition and $y.\mathcal{L}_i$ does not, and vice versa. Otherwise we iterate along $\mathcal{L}$ comparing the next condition, repeating the process until one of the solutions dominates the other or they are incomparable for every condition. In the latter case we move on to normal Pareto ranking used in NSGA-II.

In the experiments below, two conditions are used, with the first given a higher priority than the second: (1) a solution dominates another if its $f_3(\boldsymbol{S}) \leq k$ and the other's is not, and (2) a solution dominates another if its $f_3(\boldsymbol{S})$ + 0.5mm is less than the other's $f_3(\boldsymbol{S})$ value. Here k is the maximum desired value for excess stock. If these conditions are not met by either solution, the normal Pareto methods from NSGA-II are used to assign dominance. The same objective functions are used as in NSGA-II – $f_1(\boldsymbol{S})$, $f_2(\boldsymbol{S})$ and $f_3(\boldsymbol{S})$ from equations 8.4, 8.5 and 8.7 respectively.

Algorithm 8.1: Dominance assignment in Precedential Objective Ordering

```
order conditions by descending importance, giving $\mathcal{L}$
for $\mathcal{L}_i$ in $\mathcal{L}$, do
    if $x.\mathcal{L}_i$ satisfies condition $\mathcal{L}_i$ and $y.\mathcal{L}_i$ does not, do
        return $x$ dominates $y$
    else if $y.\mathcal{L}_i$ satisfies condition $\mathcal{L}_i$ and $x.\mathcal{L}_i$ does not, do
        return $y$ dominates $x$
    end if
end for
perform a normal, unbiased Pareto comparison on $x$ and $y$, as in NSGA-II
```

8.4 Results

Similarly to previous chapters, the experiments below target the problem of choosing a sequence of tools from a given library that will machine a solid into a desired shape within a predefined tolerance. The difference here is that there is an extra objective: minimising total tooling costs. The results in this chapter are divided between two main experiments. Both experiments use the same algorithms but they differ in the tool libraries employed. The first uses 12 high speed steel (HSS) flat end mill tools, with all cutting parameters derived from Machinery's Handbook (Oberg et al., 2008). This source provides reliable tool life information, meaning that an industrially relevant model can be created. The work piece has been chosen as alloy steel, ANSI 5140, with a Brinell hardness of 275 (Oberg et al., 2008). For this experiment, the algorithms are applied to Part 4, as it is a large component with many flat edges. There are 3,301 possible tool sequences using this tool library with a maximum tool sequence length of 5.

Similarly to many other research projects, such as (D'Souza et al., 2004; Wang et al., 2005a; Ahmad et al., 2010; Chen and Fu, 2011), this library includes only flat end mill tools. This means that there is a simpler relationship between tools in a sequence. Tools must be placed in size order, and the last tool in the sequence will define the final surface tolerance. Unfortunately, it proved very difficult to find similar cutting data with tool life information for other tool types. In the second group of experiments, a similar version of the tool library from Chapters 4, 5 and 7 is used. This consists of 6 flat end mill, 6 ball nosed and 6 toroidal cutters. Cutting parameters are chosen based on the recommendation of the tooling company ITC Ltd (Industrial Tool Corporation, 2012). However, tool life information - variables *C* and *n* from Taylor's Tool Life Equation - is derived from approximate values, which are suggested in (Juneja and Seth, 2003). In the second group of experiments, the algorithms are tested on parts 1, 2 and 4. Part 1 is used as it is the smallest tool, and with the given tool library it is possible to machine the whole part without replacing tools. The largest components, parts 2 and 4, are used to evaluate how well the algorithm handles replacements.

The first experiment is based on industrially relevant data but has a more restricted search space. It means that the results that the algorithms find can be trusted to provide good insights for similar industrial tasks. The second experiment increases the size and complexity of the search space, and so will test the ability of the algorithms to work on a complicated task but the actual results cannot be directly applied to real world situations, as they use approximate tool life values. However, they could still provide useful information to a decision maker.

For both groups of experiments NSGA-II (N), Weighted Objectives (WO), Guided Elitism (GE), R-NSGA-II (R) and Precedential Objective Ordering (P) are evaluated on the average hypervolume they obtain based on solutions that achieve a maximum excess stock below a specified target and calculated using $f_1(\boldsymbol{S})$ and $f_2(\boldsymbol{S})$ from equations 8.4 and 8.5.

8.4.1 Experiment 1: 12 Tool Library

8.4.1.1 Library and Data

In this first experiment, a library of 12 tools are used, and cutting data is taken from Machinery's Handbook (Oberg et al., 2008), which can be found in Table 8.3 and Table 8.4 below. Pricing data obtained from (MSC Industrial Supply Company, 2013) is also included in Table 8.3. The handbook provides cutting speed (*V*) and spindle speed (*N*) data for different tool life durations. This enables the Taylor constants *n* and *c* to be calculated in the following way:

$$V_T \, . \, T^n = C$$

$$V_{15}.\, 15^n = V_{180}.\, 180^n$$

$$\mathrm{n} = \frac{\log_{10}\left(\frac{V_{15}}{V_{180}}\right)}{\log_{10}\left(\frac{180}{15}\right)}$$

Substituting the values from Table 8.4 for a diameter of 0.125 inches:

$$\mathrm{n} = \frac{\log_{10}\left(\frac{89.23}{55.97}\right)}{\log_{10}\left(\frac{180}{15}\right)} = 0.1877 \ldots$$

Substituting the values from Table 8.4 for a diameter of 1.0 inches:

$$\mathrm{n} = \frac{\log_{10}\left(\frac{114.4}{71.76}\right)}{\log_{10}\left(\frac{180}{15}\right)} = 0.1877 \ldots$$

We find that *n* is the same for all diameter sizes. Now substituting this value for *n* into equation (8.1), to 2 decimal places, for diameters 0.125 – 0.375 inches we get:

$$C = 81.12\, X \ 45^{0.1877\ldots} = 165.73$$

and to 2 decimal places, for diameters > 0.375 we get:

$$C = 104.0 \text{ x } 45^{0.1877\ldots} = 212.48$$

We can now find the duration a tool can be usefully employed for given its cutting speed.

Table 8.4. Showing diameter (d) in inches and mm, the number of teeth (z), the feed rate per tooth in inches (f_z), Depth of Cut (DOC) in inches, Taylor constant n and Taylor constant c for the tool library used in the 12-tool experiment. Taken from (Oberg et al., 2008) and (MSC Industrial Supply Co., 2013).

d (inches)	d (mm)	z	f_z	DOC	Taylor n	Taylor c	Price (£)
0.125	3.18	4	0.001	0.05	0.188	165.732	5.91
0.25	6.35	4	0.001	0.05	0.188	165.732	6.01
0.375	9.53	4	0.001	0.05	0.188	165.732	8.78
0.5	12.70	4	0.001	0.25	0.188	212.477	8.98
0.625	15.88	4	0.001	0.25	0.188	212.477	12.5
0.75	19.05	4	0.002	0.25	0.188	212.477	25.69
0.875	22.23	4	0.002	0.25	0.188	212.477	40.72
1	25.40	6	0.002	0.25	0.188	212.477	28.5
1.125	28.58	6	0.002	0.25	0.188	212.477	38.03
1.25	31.75	6	0.002	0.25	0.188	212.477	62.24
1.375	34.93	6	0.002	0.25	0.188	212.477	71.83
1.5	38.10	6	0.002	0.25	0.188	212.477	69.43

Table 8.3. Showing the diameter (d) in inches, cutting speed (Vc) in feet per minute, spindle speed (N) in revolutions per minute and table feed (Tf) in millimetres per minute for a tool life of 15, 45 and 90 minutes for the tool library in Experiment 1, taken from (Oberg et al., 2008).

d	Vc_{15}	N_{15}	T_{f15}	Vc_{45}	N_{45}	T_{f45}	Vc_{180}	N_{180}	T_{f180}
0.125	89.23	2726.73	277.04	81.12	2478.84	251.85	55.97	1710.40	173.78
0.25	89.23	1363.36	138.52	81.12	1239.42	125.93	55.97	855.20	86.89
0.375	89.23	908.91	92.35	81.12	826.28	83.95	55.97	570.13	57.93
0.5	114.40	873.95	88.79	104.00	794.50	80.72	71.76	548.21	55.70
0.625	114.40	699.16	71.03	104.00	635.60	64.58	71.76	438.56	44.56
0.75	114.40	582.63	118.39	104.00	529.67	107.63	71.76	365.47	74.26
0.875	114.40	499.40	101.48	104.00	454.00	92.25	71.76	313.26	63.65
1	114.40	436.98	133.19	104.00	397.25	121.08	71.76	274.10	83.55
1.125	114.40	388.42	118.39	104.00	353.11	107.63	71.76	243.65	74.26
1.25	114.40	349.58	106.55	104.00	317.80	96.87	71.76	219.28	66.84
1.375	114.40	317.80	96.87	104.00	288.91	88.06	71.76	199.35	60.76
1.5	114.40	291.32	88.79	104.00	264.83	80.72	71.76	182.74	55.70

8.4.1.2 Experimental set up

The smallest two tools in the library, with diameters of 3.18mm and 6.35mm, can achieve maximum distances of excess stock of 1.2mm and 2.5mm on this component. As these are flat end mill tools, combinations of tools cannot go below these limits. Given this, the target value for excess stock, $max(d)$ as described above in equation (8.7), has been set to 1.2mm. The minimum and maximum cutting speeds are set to 1,200mm/s and 3,600mm/s respectively. Each of the 5 algorithms is tested over 1,000 trials on a cached version of the discrete (combinatorial) search space. The algorithms are tested with a budget of 200 expensive (CAM generated) tool path evaluations, and 10,000 cheaper continuous evaluations. To assess their performance, the hypervolume performance indicator is used, considering only solutions that achieve the target surface tolerance are considered.

8.4.1.3 Part 4

Figure 8.2 shows boxplots aggregating results for each algorithm over all of the 1,000 runs using Part 4 and the 12-tool library. One can immediately see that Weighted Objectives (WO) and Precedential Objective Ordering (P) perform the best on this task. Compared to the other three algorithms, they have a very little difference in hypervolume between runs. They also look almost identical in Figure 8.2. The Kruskal-Wallis one way analysis of variance by ranks was used to test the null hypothesis that independent samples from two or more algorithms come from the same distribution. This showed that there were significant differences between at least two of the algorithms. Post-hoc analysis using pairwise Wilcoxan rank sum tests showed that all of the algorithms were significantly different ($p < 0.001$) apart from P and WO ($p = 0.01$), when using the Bonferroni correction for multiple comparisons.

Median and interquartile hypervolume scores are shown in Table 8.5. Values from this table suggest that P and WO are practically identical in terms of hypervolume, only differing in minimum scores, where P has a slight edge. Figure 8.4 shows plots of the Empirical Attainment Functions (EAFs) produced by the algorithms across the different trials. P and WO have near-identical attainment surfaces, indicating how similarly they perform on this experiment.

In terms of hypervolume, the next best performing algorithm is Guided Elitism (GE). The boxplots in Figure 8.2 show that its median score is very close to P and WO, however it is not as consistent, executing a number of poorer scoring runs. Looking at the attained surfaces, we can immediately see the effect of the adaption. The attained surfaces are similar to NSGA-II (N) on the left and right sides but there is a much better convergence on a region in the middle section of the Pareto front, which it finds in almost all runs. Figure 8.5 shows the differences in EAF between selected pairs of algorithms. From this we can clearly see that GE outperforms N in this central region but is similar elsewhere. Comparing GE to WO, we also see that it discovers this particular region more frequently than WO, although the latter algorithm frequently outperforms it on the left and right sides of the front. From the boxplots, we can also see that N has a larger range of poor performing runs than GE.

Figure 8.3. Boxplots showing the distribution of hypervolumes on the y-axis obtained by the algorithms on the x-axis, applied to Part 4 with the library of 12 end mill tools over 1,000 runs (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The algorithms are NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R).

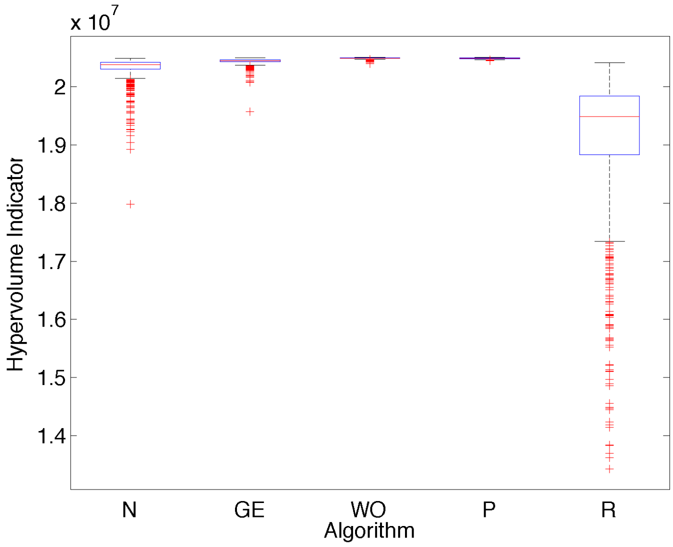


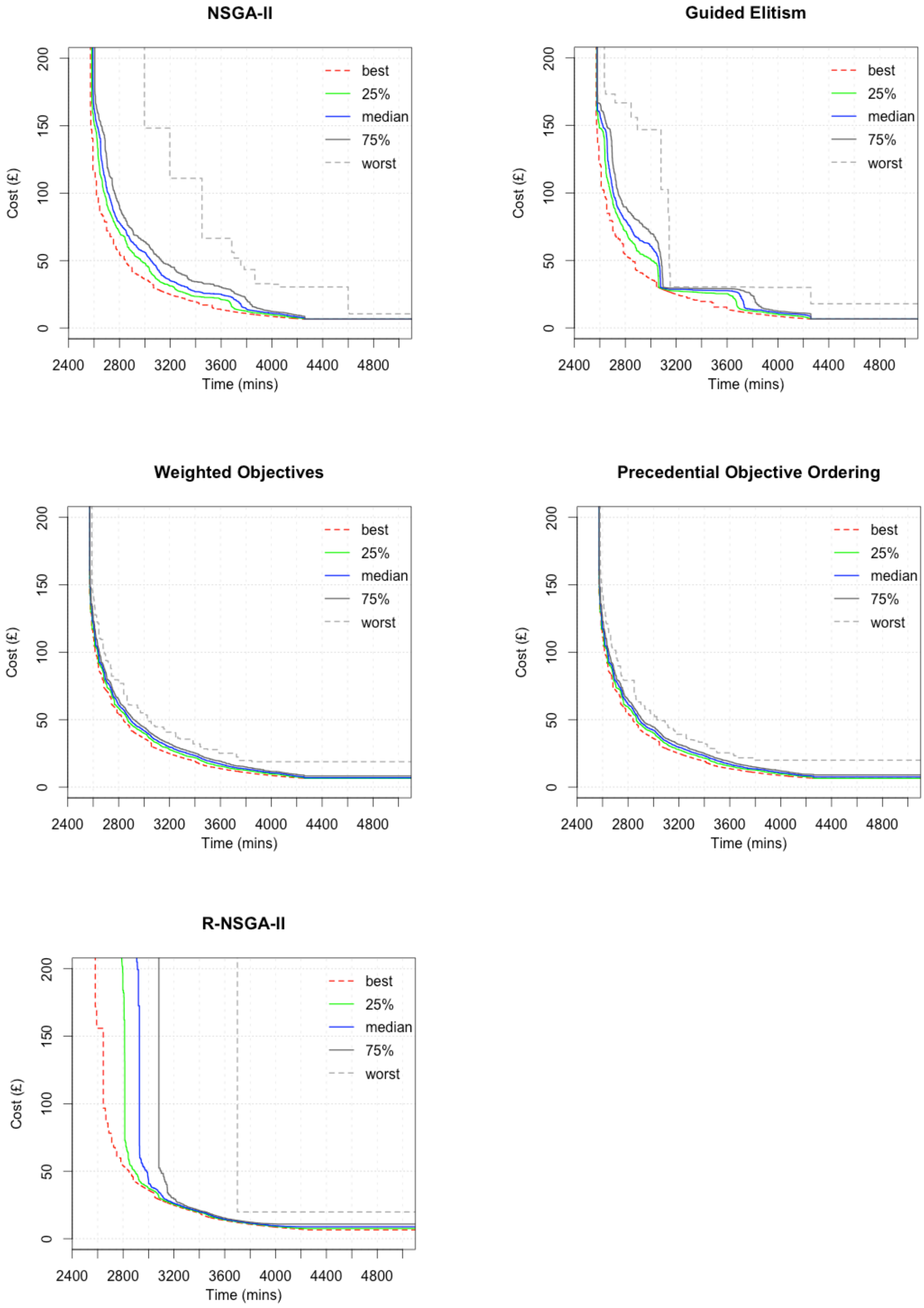
Table 8.5. Showing the hypervolumes obtained over 1,000 runs for the 5 algorithms on Part 4 with the 12-tool library. The algorithms are NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), Precedential Objective Ordering (P) and Weighted Objectives (WO). Hypervolumes are displayed as 1 x 10^4. As indicated by the superscript next to the algorithm name, all algorithms have significantly different hypervolume scores using the Kruskal-Wallis test for equal medians ($p < 1 \times 10^{-9}$), followed by a post-hoc pairwise Wilcoxan Rank Sum test for equal medians and the Bonferroni correction for multiple comparisons ($p < 1 \times 10^{-9}$), except for WO and P (p = 0.01).

Algorithm	**Min**	**1st Quart**	**Median**	**3rd Quart**	**Max**	**IQR**
N$^{(GE,R,P,WO)}$	1.798	2.031	2.038	2.042	2.049	0.012
GE$^{(N,R,P,WO)}$	1.958	2.043	2.045	2.047	2.050	0.004
R$^{(N,GE,P,WO)}$	1.342	1.884	1.948	1.984	2.042	0.100
P$^{(N,GE,R)}$	2.044	2.049	2.049	2.050	2.051	0.001
WO$^{(N,GE,R)}$	2.040	2.049	2.049	2.050	2.051	0.001

R-NSGA-II (R) has the most varied performance between different runs, and the lowest median score. From Figure 8.4, we can see that it frequently finds very good solutions in the centre and right sides of the Pareto front but discovers solutions on the left side of the front far less often. However, in the centre right region of the front R performs the best, which can be seen in the differences in EAF between R and WO, shown in Figure 8.5. This is interesting as it suggests that using preferential search can lead to more success in a particular region of the Pareto front.

In this experiment only a single tool sequence is ever employed by any of the algorithms. However, a range of different trade-off solutions for total machining time and total tooling costs are still found by varying the cutting speeds. The tool sequence used is {15.88x0.0(end mill) – 3.25x0.0(end mill)}. Figure 8.6 shows the solutions returned within the desired surface tolerance from the best performing run in terms of hypervolume by each algorithm. From this we can see that all of the algorithms produce a good approximation of the Pareto front, apart from R, which concentrates on the middle region. We can also see that GE and N produce a sparser selection of solutions. This is because not all of their returned solutions meet the excess stock requirement.

Figure 8.4. Empirical attainment functions for the five algorithms applied to Part 4 using a 12 tool library of end mills. The red dashed line shows the best solutions found over 1,000 runs. The green line shows the surface attained in the top 25% of runs. The blue line shows the surface found 50% of the time. The black solid line shows the surface attained in the 25% worst cases and the black dashed line shows the worst solutions found. Only solutions with 1.2mm of stock and under are considered.



NSGA-II

Guided Elitism

Weighted Objectives

Precedential Objective Ordering

R-NSGA-II

Figure 8.5. Showing for Part 4 with a library of 12 end mills, the differences in Empirical Attainment Function between Weighted Objectives and Guided Elitism; NSGA-II and Guided Elitism; and Weighted Objectives and R-NSGA-II. Coloured patches show the differences in attainment in favour of the algorithm named underneath the plot. White patches represent a small difference between 0% - 20% in frequency, while black patches represent large differences between 80% - 100%. On each graph, the median attained surface for the algorithm named underneath is shown with a dashed line. The best and worst surfaces found by the union of the sets returned by both algorithms are shown with solid lines.

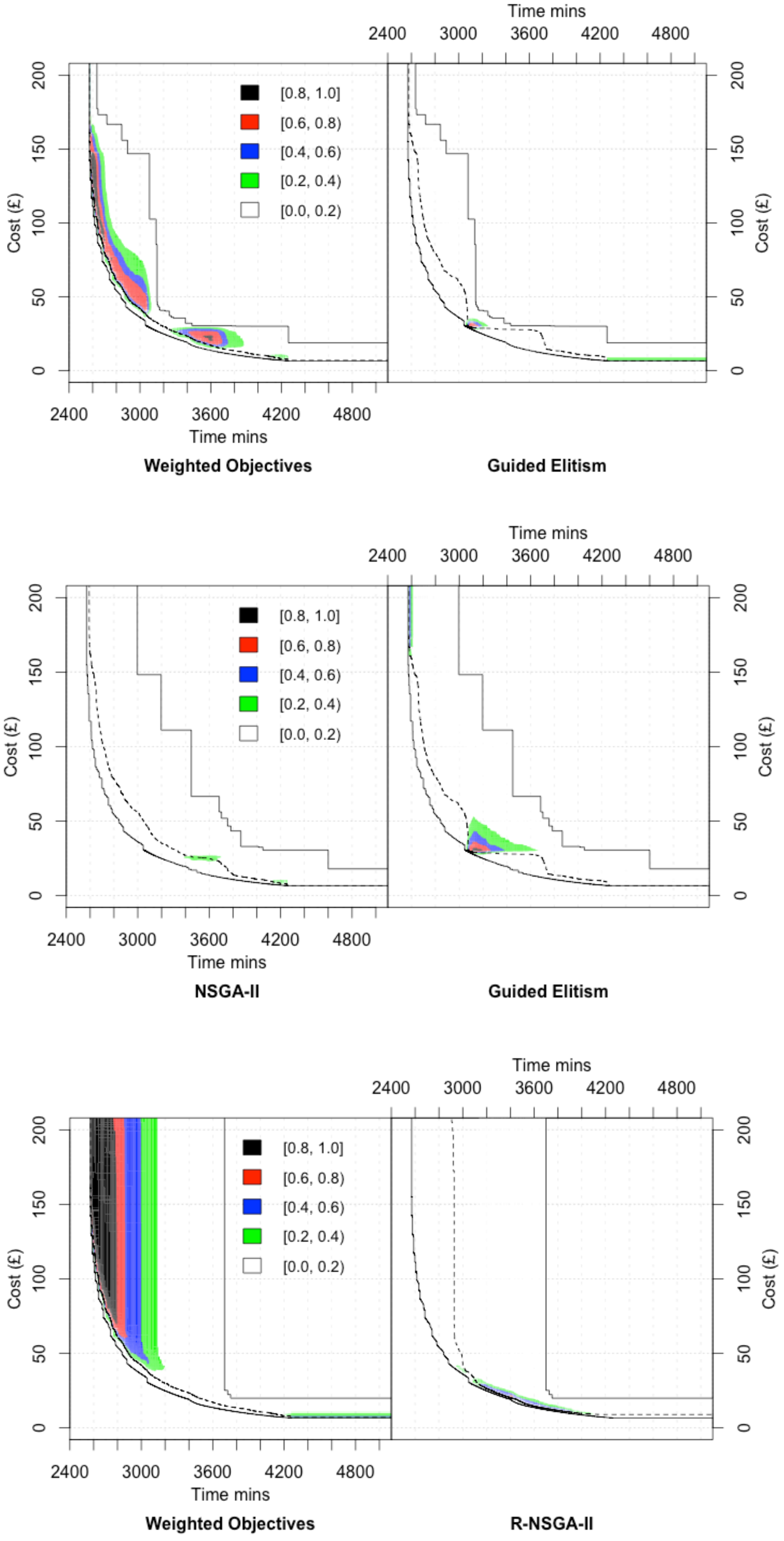
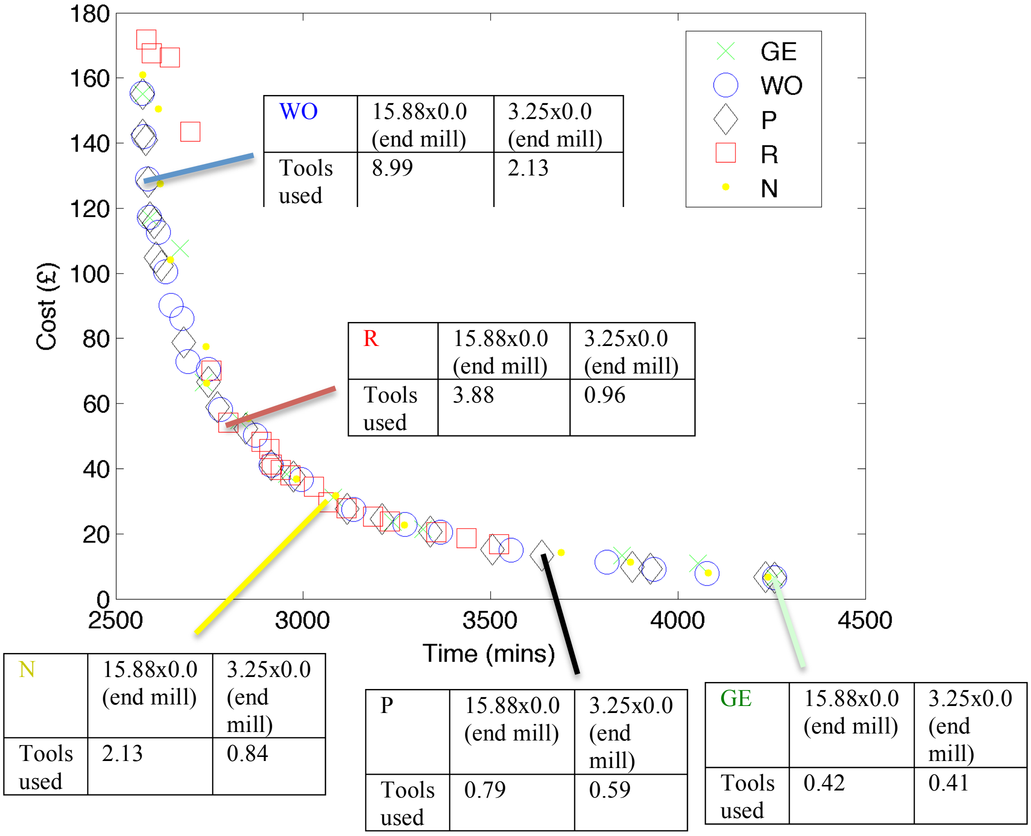


Figure 8.6. Showing the solutions found with excess material under 1mm for the highest scoring runs in terms of hypervolume for Precedential Objective Ordering (P), NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), and Weighted Objectives (WO) on Part 4 with a 12-tool library of end mills.



WO	15.88x0.0 (end mill)	3.25x0.0 (end mill)
Tools used	8.99	2.13

R	15.88x0.0 (end mill)	3.25x0.0 (end mill)
Tools used	3.88	0.96

N	15.88x0.0 (end mill)	3.25x0.0 (end mill)
Tools used	2.13	0.84

P	15.88x0.0 (end mill)	3.25x0.0 (end mill)
Tools used	0.79	0.59

GE	15.88x0.0 (end mill)	3.25x0.0 (end mill)
Tools used	0.42	0.41

8.4.2 Experiment 2: 18-Tool Library

In the first experiment, which employed a 12-tool library of end mills, we have seen that multi-objective algorithms work well in a small but real world search space. We now move on to a much larger and more complex search space but use estimated Taylor Tool Life constants. This allows us to evaluate how feasible the approach is in a more complex domain.

This group of experiments uses an 18-tool library of mixed tool types. We first describe the tool library and cutting data before analysing the results of the algorithms on parts 1, 2 and 4 individually. These parts were chosen because they have different features. Part 1 is small and so it should be possible to not have to replace tools using this library. Part 2 is larger, so the algorithms will have to handle tool replacements. Part 4 is also large but only a subset of the ball nose tools can reach the desired surface tolerance, meaning that there are fewer feasible solutions.

8.4.2.1 Library and Cutting Data

For these groups of experiments, tool data was obtained from the Industrial Tooling Corporation catalogue (ITC Ltd., 2012). As in Chapters 4, 5 and 7, 18 tools are used in total, 3 sets of 6 toroidal, flat end mill and ball nosed tools. Tools are made of solid carbide and were chosen from the catalogue based on their ability to machine a work piece made of ANSI 5140 alloy steel. Although the tooling

Table 8.6. Showing diameter (d) in mm, the number of teeth (z), Depth of Cut (DOC) in mm, Taylor constant *n* and Taylor constant *c* for the tool library used in the 18-tool library experiments. Data taken from (ITC Ltd, 2012).

d	z	Tool Type	Corner Radius	DOC	Taylor n	Taylor c	Price (£)
6	4	End mill	0	1.5	0.3	513.95	42.48
8	4	End mill	0	2.0	0.3	513.95	57.89
10	4	End mill	0	2.5	0.3	513.95	84.38
12	4	End mill	0	3.0	0.3	513.95	110.3
16	4	End mill	0	4.0	0.3	513.95	179.03
20	4	End mill	0	5.0	0.3	513.95	299.04
6	4	Toroidal	0.5	0.5	0.3	462.55	40.1
8	4	Toroidal	1.0	1.0	0.3	462.55	51.85
10	4	Toroidal	1.0	1.0	0.3	462.55	65.78
12	4	Toroidal	2.0	2.0	0.3	462.55	88.87
16	4	Toroidal	2.0	2.0	0.3	462.55	151.86
20	4	Toroidal	3.0	3.0	0.3	462.55	246.49
6	4	Ball Nose	3.0	0.6	0.3	185.02	36.23
8	4	Ball Nose	4.0	0.8	0.3	185.02	42.24
10	4	Ball Nose	5.0	1.0	0.3	185.02	64.8
12	4	Ball Nose	6.0	1.2	0.3	185.02	94.29
16	4	Ball Nose	8.0	1.6	0.3	185.02	147.77
20	4	Ball Nose	10.0	2.0	0.3	185.02	259.8

company provides cutting data information, we were unable to obtain real world Taylor Tool Life parameters. To compensate for this, the Taylor *n* parameter has been set to 0.3, based on recommendations for carbide tools taken from (Juneja and Seth, 2003). Taylor *C* was calculated by assuming that the manufacturer's recommendations provided a tool life of 45 minutes and using equation (8.1) to derive *c*. Data for the tool library is presented in Table 8.6 and Table 8.7.

8.4.2.2 Experimental Setup

As with the 12-tool experiment, for each experiment below, each of the 5 algorithms has been tested over 1,000 runs on a cached version of the discrete (combinatorial) search space. The algorithms were tested with an evaluations budget of 500 expensive (CAM-generated) tool path evaluations, and 10,000 cheaper tool life calculation evaluations. Similarly to the previous experiment, the hypervolume performance indicator is used to assess the algorithms. Only solutions that achieve the target surface tolerance are considered. The minimum and maximum cutting speeds are set to being

Table 8.7. Showing the diameter (d) in mm, tool type, corner radius in mm, number of teeth (z), feed rate per tooth (f_z) in mm, cutting velocity (Vc) in feet per minute, spindle speed (N) in revolutions per minute and table feed (Tf) in millimetres per minute for a tool life of 45 minutes, for the 18-tool library experiments.

d	Tool Type	Corner Radius	z	F_z	Vc_{45}	N_{45}	Tf_{45}
6	End mill	0	4	0.03	196.85	3183.10	381.97
8	End mill	0	4	0.04	196.85	2387.32	381.97
10	End mill	0	4	0.045	196.85	1909.86	343.77
12	End mill	0	4	0.05	196.85	1591.55	318.31
16	End mill	0	4	0.055	196.85	1193.66	262.61
20	End mill	0	4	0.06	196.85	954.93	229.18
6	Toroidal	0.5	4	0.04	213.25	3448.36	551.74
8	Toroidal	1.0	4	0.043	213.25	2586.27	444.84
10	Toroidal	1.0	4	0.047	213.25	2069.01	388.97
12	Toroidal	2.0	4	0.05	213.25	1724.18	344.84
16	Toroidal	2.0	4	0.075	213.25	1293.13	387.94
20	Toroidal	3.0	4	0.1	213.25	1034.51	413.80
6	Ball Nose	3.0	4	0.013	164.04	2652.58	137.93
8	Ball Nose	4.0	4	0.017	164.04	1989.44	135.28
10	Ball Nose	5.0	4	0.02	164.04	1591.55	127.32
12	Ball Nose	6.0	4	0.025	164.04	1326.29	132.63
16	Ball Nose	8.0	4	0.05	164.04	994.72	198.94
20	Ball Nose	10.0	4	0.075	164.04	795.77	238.73

between (3,600mm/s – 8,400mm/s) for end mill tools, (2,800mm/s – 7,000mm/s) for ball nose tools and (4,000mm/s – 9,000 mm/s) for toroidal tools.

8.4.2.3 Part 1

Part 1 is the smallest of the three parts tested with the 18-tool library of mixed tool types. The results, presented in Table 8.8, show some interesting differences from the 12-tool experiment above. Figure 8.7 presents boxplots showing the distribution of hypervolume scores achieved by the algorithms over the 1,000 trials. The most striking feature is that WO performs much worse than it did in the experiment above, and is considerably worse than the other algorithms. Figure 8.8 shows the attained surfaces for each of the algorithms. We can see here that WO appears to converge on a suboptimal Pareto front. By this we mean that it has found a diverse spread of solutions but that these are dominated by the solutions returned by the other algorithms. Figure 8.9 shows differences in EAFs for selected pairs of algorithms. Comparing WO to P, we see that there is a large distance between the median fronts of the two algorithms, and a large area that is not attained by WO. The reason for this difference is illuminated in Figure 8.10, showing the best runs for each algorithm, where we see that WO is the only algorithm to use the sequence {10.0x0.0(end mill) – 6.0x0.5(toroidal) – 6.0x3.0(ballnose)}.

Figure 8.7. Boxplots showing the distribution of hypervolumes on the y-axis obtained by the algorithms on the x-axis, applied to Part 1 with the library of 18 mixed tool types over 1,000 runs (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The algorithms are NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R).

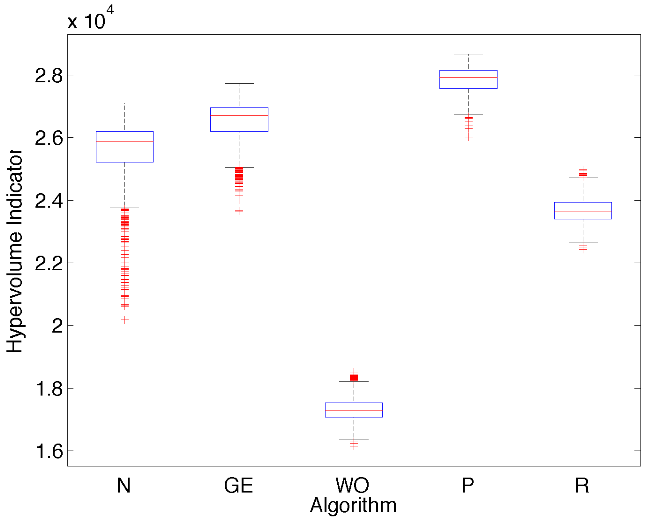


Table 8.8. Showing the hypervolumes obtained over 1,000 runs for the 5 algorithms on Part 1 with the 18 tool libray. The algorithms are NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), Precedential Objective Ordering (P) and Weighted Objectives (WO). Hypervolumes are displayed as 1 x 10^4. As indicated by the subscript next to the algorithm name, all algorithms have significantly different hypervolume scores using the Kruskal-Wallis test for equal medians ($p < 1 \times 10^{-9}$), followed by a post-hoc pairwise Wilcoxan Rank Sum test for equal medians and the Bonferroni correction for multiple comparisons ($p < 1 \times 10^{-9}$).

Algorithm	**Min**	**1st Quart**	**Median**	**3rd Quart**	**Max**	**IQR**
N$^{(GE,R,P,WO)}$	2.018	2.522	2.586	2.620	2.711	0.098
GE$^{(N,R,P,WO)}$	2.366	2.619	2.671	2.695	2.773	0.076
R$^{(N,GE,P,WO)}$	2.243	2.340	2.365	2.394	2.498	0.054
P$^{(N,GE,R,WO)}$	2.602	2.757	2.792	2.815	2.867	0.058
WO$^{(N,GE,R,P)}$	1.615	1.707	1.729	1.754	1.852	0.047

The best performing algorithm in terms of average hypervolume is P. The attained surfaces show that it finds a diverse range of solutions, and is especially consistent at locating optimal solutions, in terms of the best it can find, in the centre region of the front, which offers the best trade-off solutions.

Compared to the result in the first experiment, there appears to be more varied hypervolume scores over the 1,000 trials. P consistently reaches its best attained surface in the middle section of the front but on the right and left sides of the front, the median attained surface is a small amount of distance away from the best found. R finds a much narrower selection of solutions than the other algorithms, concentrating on the central region of the front. Comparing R to P in Figure 8.9, we see that in this central region, these algorithms perform virtually identically. R is extremely consistent in this region and even in its worst case it reaches a point on the best-found surface. However, it is vastly outperformed on the right side of the front, finding almost no solutions.

GE and N both perform well on this part. Unlike WO, they both consistently reach points on the front located by P. GE has a higher median hypervolume score than N and a lower interquartile range. Comparing the EAFs of the two in Figure 8.9, we see that there is a particular region in the centre but

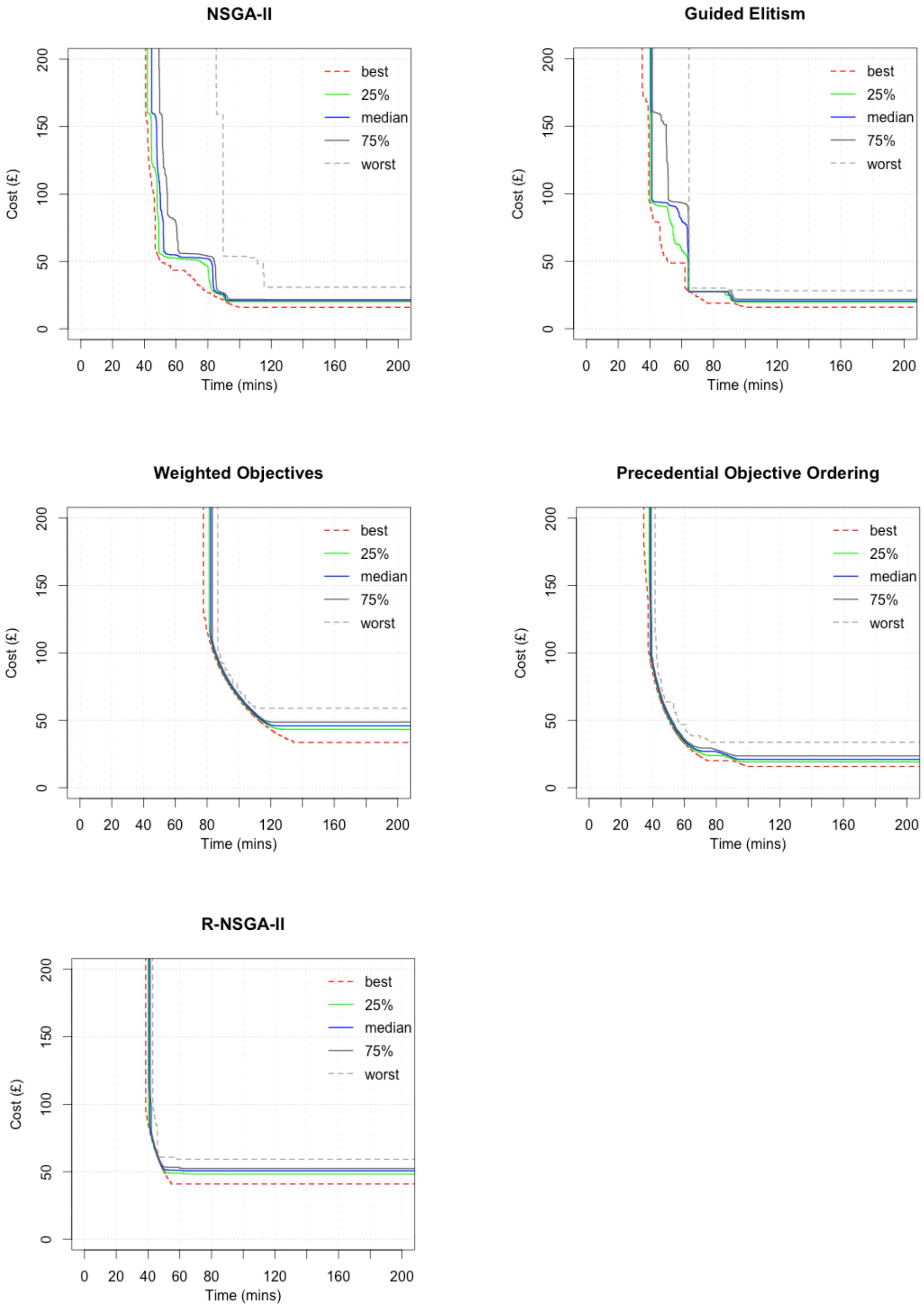
Table 8.9. Showing the unique tool sequences used in solutions by NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R) on their individual highest scoring run according to the hypervolume indicator on Part 1 with an 18-tool library of mixed tool types.

Tool Sequence	Algorithm
8.0x1.0(toroidal)	GE
8.0x0.0(end mill) – 8.0x1.0(toroidal)	N
8.0x0.0(end mill) – 6.0x0.5(toroidal)	N,P
20.0x0.0(end mill) – 8.0x1.0(toroidal)	N,P
10.0x0.0(end mill) – 8.0x1.0(toroidal)	GE,N,P,R
10.0x0.0(end mill) – 6.0x0.5(toroidal)	N
10.0x0.0(end mill) – 6.0x0.5(toroidal) – 6.0x3.0(ballnose)	WO
8.0x0.0(end mill) – 8.0x1.0(toroidal)	N

towards the cheaper and more time consuming solutions that GE reaches every time and N rarely finds. GE also more frequently finds better solutions towards the top left of the front. However, N locates a central region much more frequently than GE.

Figure 8.10 shows the solutions from the highest scoring run for each algorithm. On these runs, 8 unique tool sequences are used, which are presented in Table 8.9. N uses the most, with 4 different sequences, although some of the solutions found are dominated. WO uses only a single 3-tool sequence, and all of these solutions are dominated by some distance by the other algorithms. The most successful algorithm is clearly P, which uses 3 separate sequences to cover a wide range of high quality solutions. From this figure we can also see that R congregates exclusively on the central region of the front but finds very good trade-off solutions. It is interesting to see that different tool sequences provide access to different regions in the Pareto front. For example, P uses {20.0x0.0(end mill) – 8.0x1.0(toroidal)}, starting with a large, expensive tool, to find a very fast but expensive solution. Conversely it uses smaller tools in {8.0x0.0(end mill) – 6.0x0.5(toroidal)} to obtain a cheaper but slower solution.

Figure 8.8. Empirical attainment functions for the five algorithms applied to Part 1 using an 18 tool library. The red dashed line shows the best solutions found over 1,000 runs. The green line shows the surface attained in the top 25% of runs. The blue line shows the surface found 50% of the time. The black solid line shows the surface attained in the 25% worst cases and the black dashed line shows the worst solutions found. Only solutions with 1.0mm of stock and under are considered.



NSGA-II

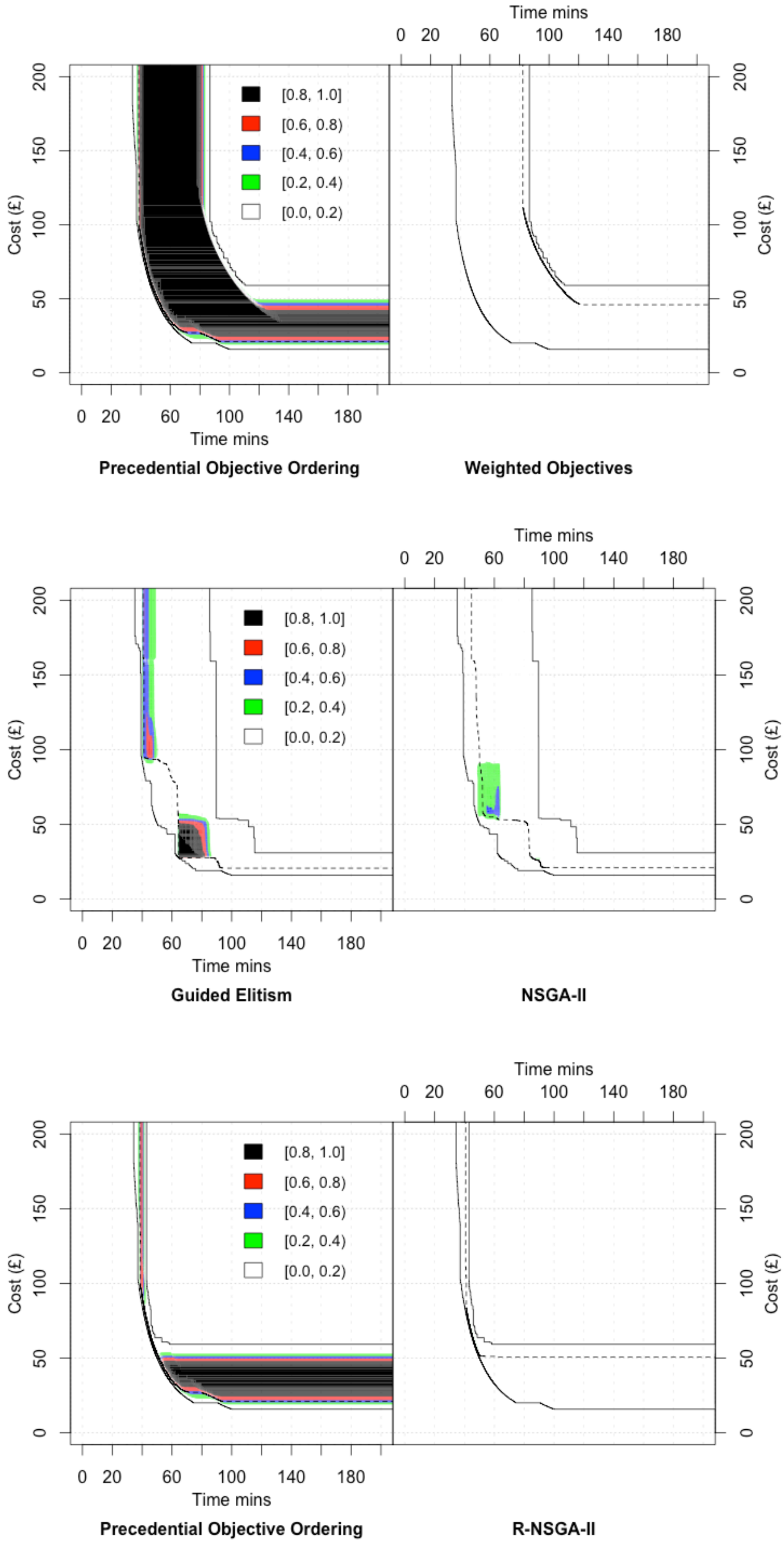
Guided Elitism

Weighted Objectives

Precedential Objective Ordering

R-NSGA-II

Figure 8.9. Showing for Part 1 with a library of 18 tools, the differences in Empirical Attainment Function between Precedential Objective Ordering and Weighted Objectives; NSGA-II and Guided Elitism; and Precedential Objective Ordering and R-NSGA-II. Coloured patches show the differences in attainment in favour of the algorithm named underneath the plot. White patches represent a small difference between 0% - 20% in frequency, while black patches represent large differences between 80% - 100%. On each graph, the median attained surface for the algorithm named underneath is shown with a dashed line. The best and worst surfaces found by the union of the sets returned by both algorithms are shown with solid lines.



Precedential Objective Ordering

Weighted Objectives

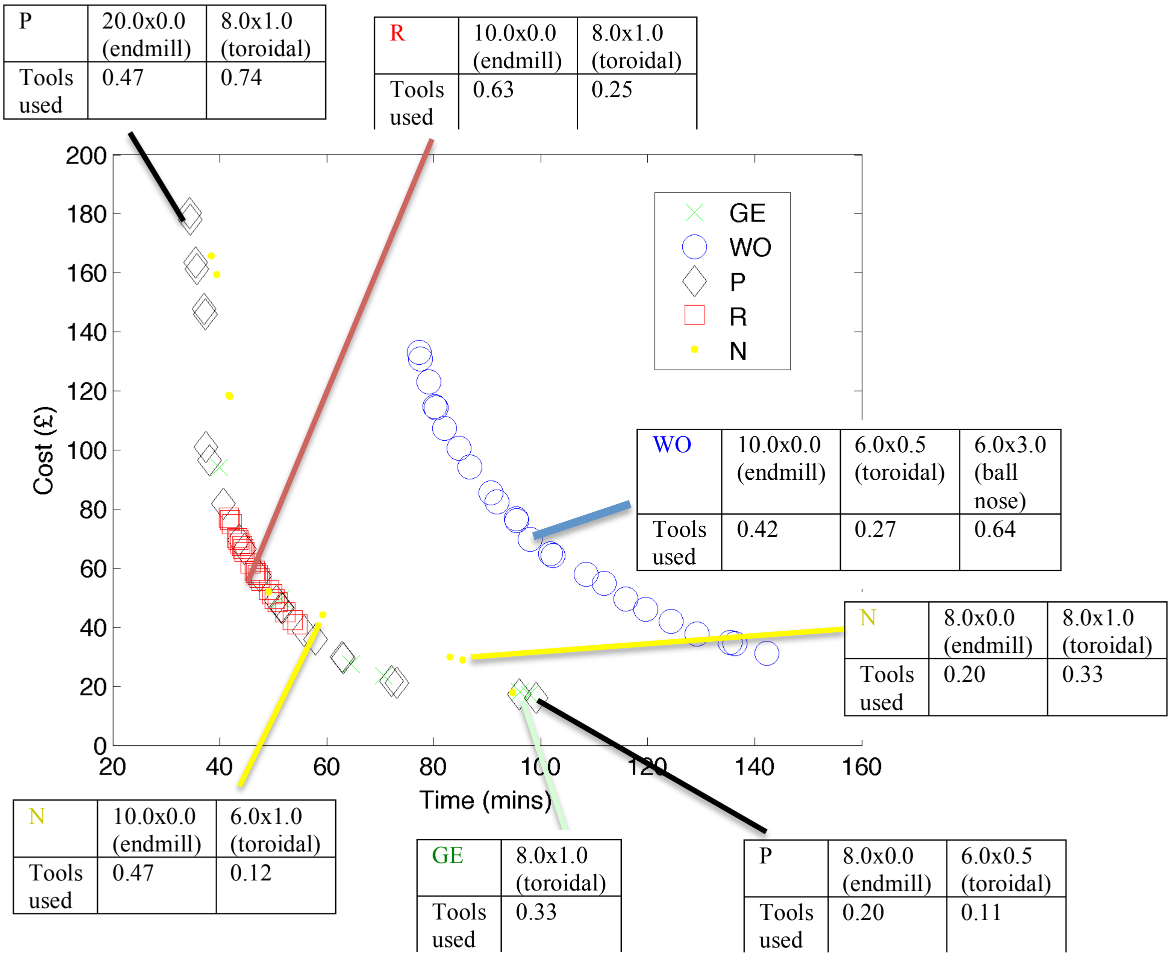
Guided Elitism

NSGA-II

Precedential Objective Ordering

R-NSGA-II

Figure 8.10. Showing the solutions found with excess material under 1mm for the highest scoring runs in terms of hypervolume for Precedential Objective Ordering (P), NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), and Weighted Objectives (WO) on Part 1 with the 18 tool library of mixed tool types.



P	20.0x0.0 (endmill)	8.0x1.0 (toroidal)
Tools used	0.47	0.74

R	10.0x0.0 (endmill)	8.0x1.0 (toroidal)
Tools used	0.63	0.25

WO	10.0x0.0 (endmill)	6.0x0.5 (toroidal)	6.0x3.0 (ball nose)
Tools used	0.42	0.27	0.64

N	8.0x0.0 (endmill)	8.0x1.0 (toroidal)
Tools used	0.20	0.33

N	10.0x0.0 (endmill)	6.0x1.0 (toroidal)
Tools used	0.47	0.12

GE	8.0x1.0 (toroidal)
Tools used	0.33

P	8.0x0.0 (endmill)	6.0x0.5 (toroidal)
Tools used	0.20	0.11

Figure 8.11. Boxplots showing the distribution of hypervolumes on the y-axis achieved by the algorithms on the x-axis, applied to Part 2 with the 18 tool library of mixed tool types over 1,000 runs (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The algorithms are NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R).

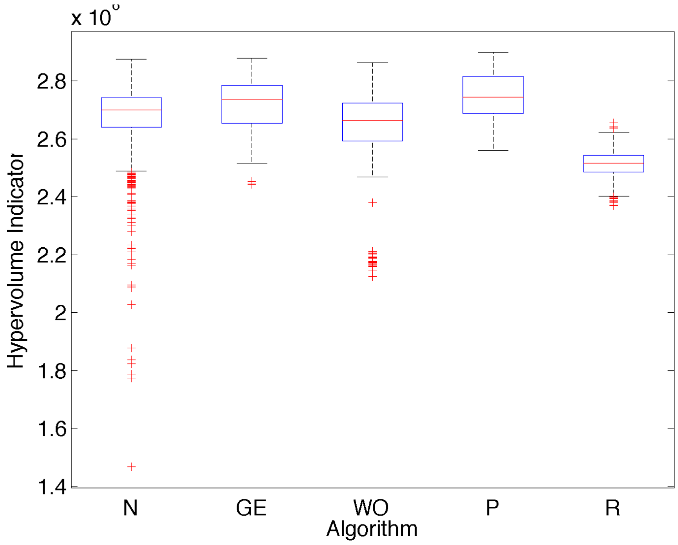


Table 8.10. Showing the hypervolumes achieved over 1000 runs for the 5 algorithms on Part 2 with the 18 tool library. The algorithms are NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), Precedential Objective Ordering (P) and Weighted Objectives (WO). Hypervolumes are displayed as 1 x 10^6. As indicated by the subscript next to the algorithm name, all algorithms have significantly different hypervolume scores using the Kruskal-Wallis test for equal medians ($p < 1 \times 10^{-9}$), followed by a post-hoc pairwise Wilcaxon Rank Sum test for equal medians and the Bonferroni correction for multiple comparisons ($p < 1 \times 10^{-9}$).

Algorithm	Min	1st Quart	Median	3rd Quart	Max	IQR
$\mathbf{N}^{(GE,R,P,WO)}$	1.467	2.642	2.701	2.743	2.876	0.102
$\mathbf{GE}^{(N,R,P,WO)}$	2.444	2.654	2.736	2.785	2.879	0.131
$\mathbf{R}^{(N,GE,P,WO)}$	2.370	2.487	2.516	2.543	2.656	0.057
$\mathbf{P}^{(N,GE,R,WO)}$	2.560	2.689	2.745	2.817	2.900	0.128
$\mathbf{WO}^{(N,GE,R,P)}$	2.124	2.593	2.664	2.724	2.865	0.131

8.4.2.4 Part 2

Part 2 is much larger than Part 1. The boxplot in Figure 8.11, showing the distributions of hypervolume scores obtained, suggests that the algorithms are a lot more even in terms of hypervolume than with Part 1. However, there are marked differences between the algorithms in terms of which regions of the Pareto front are concentrated on.

P is once again the best performing algorithm in terms of average hypervolume. However, looking at the attained surfaces in Figure 8.12, it is clear that it much more frequently finds good solutions on the left side of the Pareto front than the centre and right sides. Conversely, GE and N frequently converge on the left and right sides of the Pareto front. GE also very frequently converges on a specific point on the bottom right of the central region. Comparing the differences in EAF between GE and N in Figure 8.13, we see that N does not return solutions in this central region but finds very good solutions more frequently than GE on the left and right sides of the front. Comparing GE to P, we see that P dominates greatly on the left side, especially in the majority of the central region, but that GE dominates on the right side. The surfaces attained by WO look similar to those attained by P, concentrating on very good solutions on the left side of the Pareto front, although it does not find as good solutions in the central region. Interestingly, it finds better solutions on the far right of the front, the cheaper solutions with longer machining times, although these are regularly found or dominated by GE.

R largely concentrates solutions on a small section in the middle of the Pareto front, skewed slightly towards faster, more expensive solutions. Similarly to Part 1, these solutions are consistently found and are located on the surface of the best solutions found by any of the algorithms in this region. However, as the algorithm does not find solutions towards the bottom right of the front, it does not achieve as high of a hypervolume score as the others.

Figure 8.14 shows the solutions found with maximum excess stock under 1.0mm during the run scoring highest in hypervolume for GE, WO, P and R. In their highest scoring runs, GE, N (which is not included in the figure) and R all use solutions consisting of two tool sequences:

{12.0x0.0(end mill) – 8.0x1.0(toroidal)}

{20.0x0.0(end mill) – 12.0x0.0(end mill) – 8.0x1.0(toroidal)}

P shares one of these tool sequences and uses another:

{12.0x0.0(end mill) – 8.0x1.0(toroidal)}

{10.0x0.0(end mill) – 8.0x1.0(toroidal)}

and WO uses three different tool sequences:

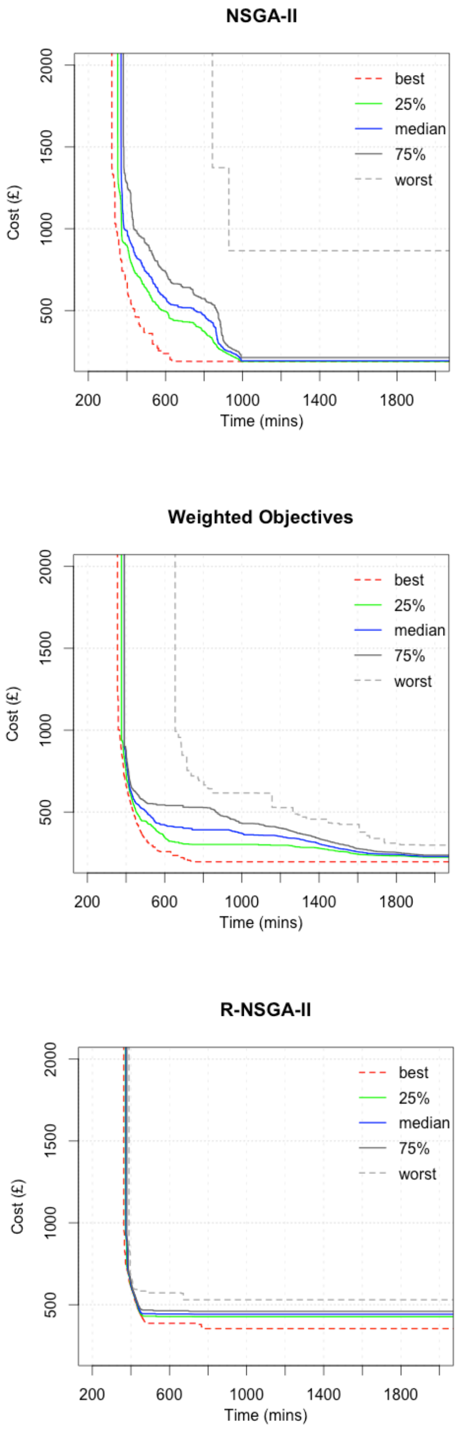
{20.0x0.0(end mill) – 6.0x0.5(toroidal)}

{12.0x0.0(end mill) – 6.0x0.5(toroidal)}

{12.0x0.0(end mill) – 6.0x0.5(toroidal) – 8.0x4.0(ball nose) – 6.0x3.0(ball nose)}

Again we see that different tool sequences can be used to obtain different parts of the Pareto front. For example, P uses {12.0x0.0(end mill) – 8.0x1.0(toroidal)} on the centre of the Pareto front for faster solutions and {10.0x0.0(end mill) – 8.0x1.0(toroidal)} towards the right of the Pareto front for slower but cheaper solutions. WO uses a number of different tool sequence strategies but they appear to mainly be dominated by other solutions, especially the four-tool sequence.

Figure 8.12. Empirical attainment functions for the five algorithms applied to Part 2 using an 18 tool library. The red dashed line shows the best solutions found over 1,000 runs. The green line shows the surface attained in the top 25% of runs. The blue line shows the surface found 50% of the time. The black solid line shows the surface attained in the 25% worst cases and the black dashed line shows the worst solutions found. Only solutions with 1.0mm of stock and under are considered.


NSGA-II

GE-NSGA-II

Weighted Objectives

Precedential Objective Ordering

R-NSGA-II

Figure 8.13. Showing for Part 2 with a library of 18 tools, the differences in Empirical Attainment Function between Precedential Objective Ordering and Guided Elitism; NSGA-II and Guided Elitism; Precedential Objective Ordering and Weighted Objectives; and Guided Elitism and Weighted Objectives. Coloured patches show the differences in attainment in favour of the algorithm named underneath the plot. White patches represent a small difference between 0% - 20% in frequency, while black patches represent large differences between 80% - 100%. On each graph, the median attained surface for the algorithm named underneath is shown with a dashed line. The best and worst surfaces found by the union of the sets returned by both algorithms are shown with solid lines.

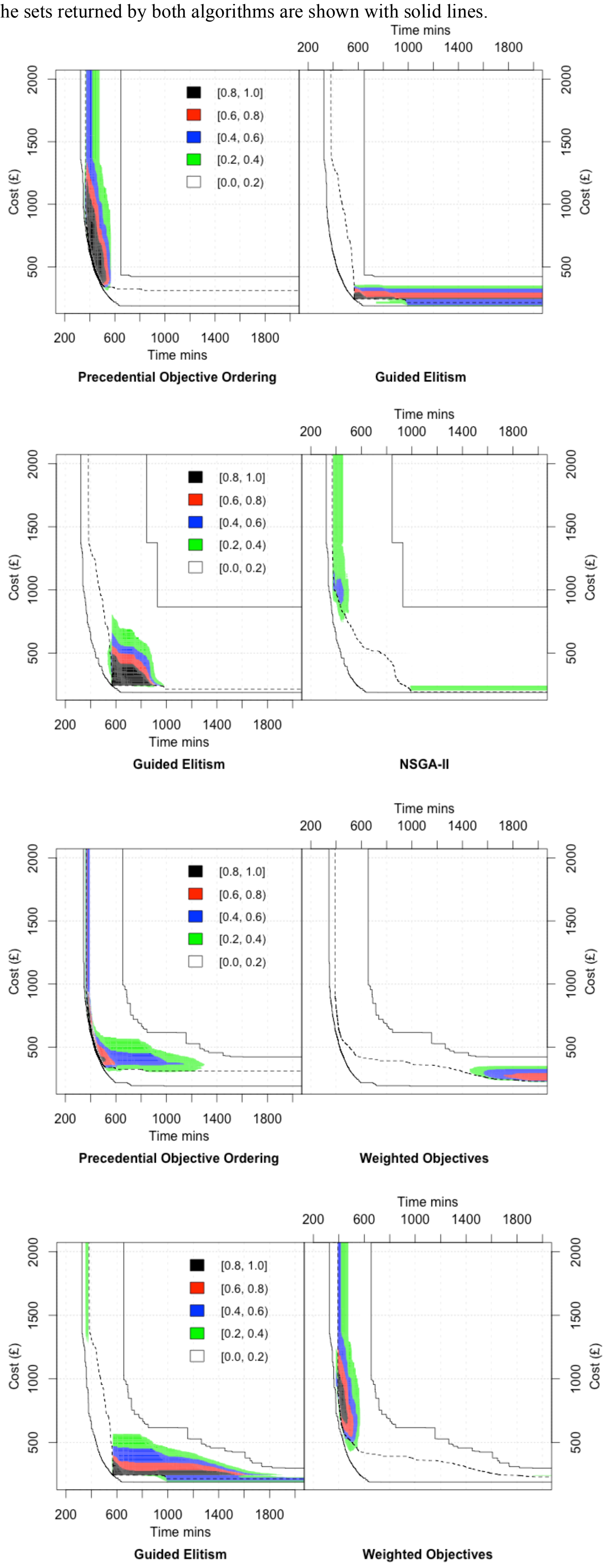
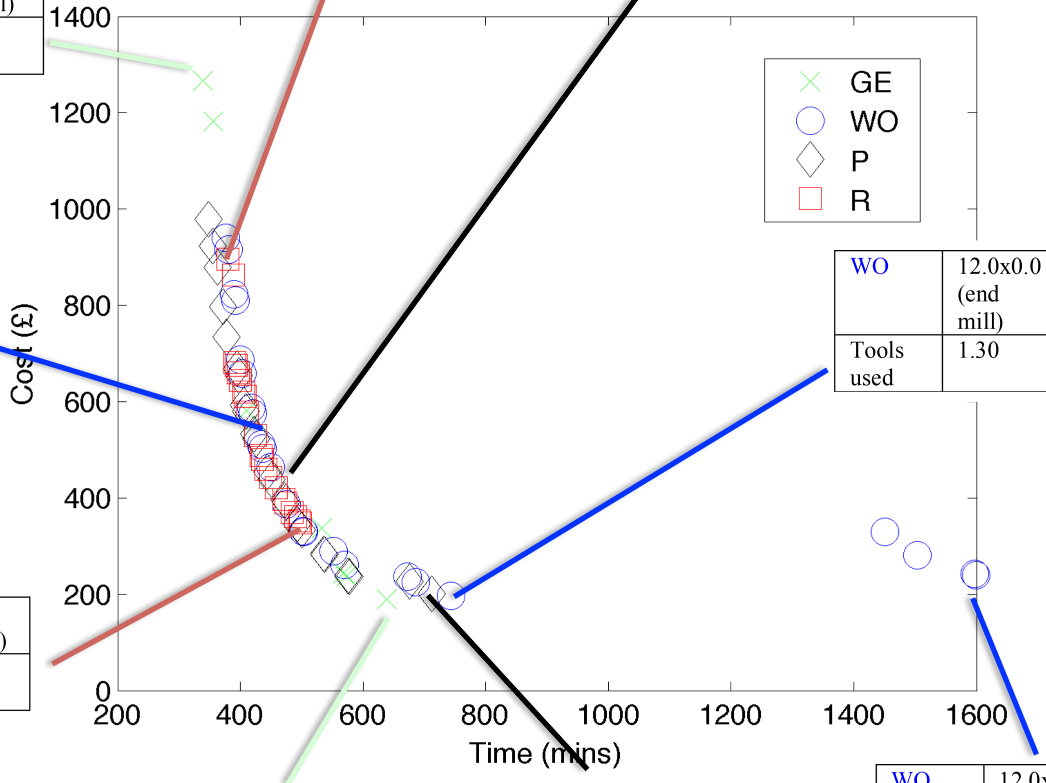


Figure 8.14. Showing the solutions found with excess material under 1mm for the highest scoring runs in terms of hypervolume for Precedential Objective Ordering (P), Guided Elitism (GE), R-NSGA-II (R), and Weighted Objectives (WO) on Part 2 with the 18 tool library of mixed tool types.



GE	20.0x0.0 (end mill)	12.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	2.91	0.95	3.96

R	20.0x0.0 (end mill)	12.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	1.99	1.32	3.13

P	12.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	2.76	2.87

WO	20.0x0.0 (end mill)	6.0x0.5 (toroidal)
Tools used	1.59	4.84

WO	12.0x0.0 (end mill)	6.0x0.5 (toroidal)
Tools used	1.30	1.28

R	12.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	2.16	2.36

GE	12.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	1.31	0.86

P	10.0x0.0 (end mill)	8.0x1.0 (toroidal)
Tools used	2.76	2.87

WO	12.0x0.0 (end mill)	6.0x0.5 (toroidal)	8.0x4.0 (ball nose)	6.0x3.0 (ball nose)
Tools used	1.31	1.29	0.57	0.54

Figure 8.15. Boxplots showing the distribution of hypervolumes on the y-axis achieved by the algorithms on the x-axis, applied to Part 4 with the 18 tool library of mixed tool types over 1,000 runs (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). The algorithms are NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R).

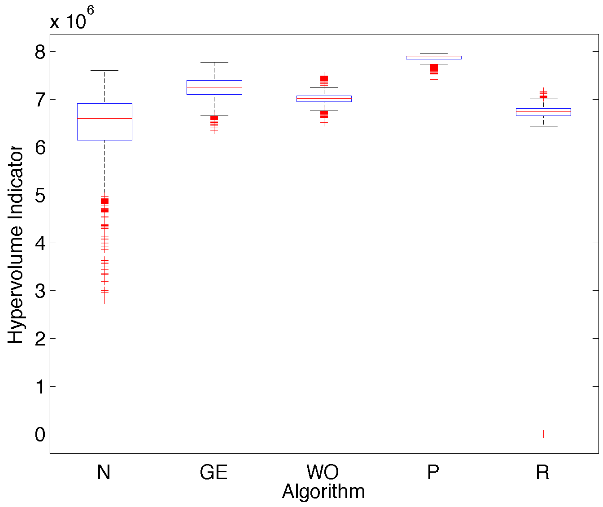


Table 8.11. Showing the hypervolumes achieved over 1000 runs for the 5 algorithms on Part 4 with the 18 tool library. The algorithms are NSGA-II (N), Guided Elitism (GE), R-NSGA-II (R), Precedential Objective Ordering (P) and Weighted Objectives (WO). Hypervolumes are displayed as 1 x 10^6. As indicated by the subscript next to the algorithm name, all algorithms have significantly different hypervolume scores using the Kruskal-Wallis test for equal medians ($p < 1 \times 10^{-9}$), followed by a post-hoc pairwise Wilcoxan Rank Sum test for equal medians and the Bonferroni correction for multiple comparisons ($p < 1 \times 10^{-9}$).

Algorithm	Min	1st Quart	Median	3rd Quart	Max	IQR
$\mathbf{N}^{(GE,R,P,WO)}$	2.802	6.143	6.599	6.912	7.599	0.769
$\mathbf{GE}^{(N,R,P,WO)}$	6.351	7.097	7.253	7.398	7.769	0.301
$\mathbf{R}^{(N,GE,P,WO)}$	0.000	6.661	6.739	6.808	7.164	0.147
$\mathbf{P}^{(N,GE,R,WO)}$	7.411	7.841	7.884	7.910	7.966	0.069
$\mathbf{WO}^{(N,GE,R,P)}$	6.510	6.950	7.020	7.077	7.490	0.127

8.4.2.5 Part 4

Part 4 is interesting because with the chosen tool library there are far fewer tool sequences that can achieve a maximum excess stock of 1mm or less than on parts 1 and 2. This is because only the ball nose tools can get within this surface tolerance. Looking at Figure 8.15, which shows boxplots of the distribution of hypervolume scores obtained, we see that in this experiment, N is now the worst performing algorithm on average. It has a large number of poor performing runs and from Table 8.11 we can see that it has a much lower 1st quartile score than the other algorithms. The fact that the other algorithms (apart from R on one occasion) find much higher scoring solutions in the worst cases, indicates that the use of preferential search (in R and GE) or constraint handling techniques (in P and WO) helps find a more diverse spread of desired solutions on this search space.

As with the 18 tool experiments on parts 1 and 2, P is the best performing algorithm in terms of hypervolume. It has the highest median hypervolume and the lowest interquartile range. In the attained surfaces shown in Figure 8.16 we can see that P consistently gets very close to its best-found surface, although we can see that the largest distance from the best and median attained surfaces occurs in the central region of the Pareto front. Comparing the differences in EAFs in Figure 8.17, we see that while P clearly outperforms the other algorithms in finding a diverse range of quality solutions, there are particular points in the central region where both R and GE frequently find better solutions.

WO also finds a diverse spread of solutions but the differences in EAF show that P vastly outperforms it in terms of the Pareto front it obtains. There is a large distance between the median attained surfaces of P and WO. This implies that again, as with Part 1, WO converges on a suboptimal front, and in Figure 8.18, which shows the solutions found by the run scoring the highest in terms of hypervolume, we see that the other algorithms dominate all of the solutions found by WO.

The inclusion of preferential search in GE has clear benefits over N in terms of obtaining consistently good hypervolume. The main differences in the attained surfaces of the two algorithms, seen in Figure 8.17, is a much more frequent convergence on good solutions in the central region of the Pareto front by GE. Both algorithms converge frequently on very good solutions on the right hand side of the Pareto front, and in the differences in EAFs shown in Figure 8.17, we can see that GE achieves better performance than P in this area. However, the algorithm is clearly outperformed in the left region of the front.

Similarly to the experiments on parts 1 and 2, R finds solutions in a narrower region than the other algorithms. This is clear in the highest scoring run seen in Figure 8.18 and the attained surfaces seen in Figure 8.16. However, comparing EAFs we see that in this region it performs very well and at points better than P. On one run R does not find any solutions within the surface tolerance requirement. While this is an isolated case, it could imply that more importance needs to be levied on the excess stock value on the reference point. In the experiments in this chapter this has been set to a weighting of 10, while the other two objectives have been given a weighting of 1. On a search space where there are fewer feasible solutions, it may be that a higher weighting is needed in order to converge to within the desired region.

The feasible solutions found by GE, WO, P and R from their highest scoring run are shown in Figure 8.18. Table 8.12 shows all of the unique tool sequences that form these solutions and the algorithms that use them. Different tool sequences are used to obtain different parts of the Pareto front. For example, R uses two different tool sequences to find solutions on the left and right sides of the central region. Both P and GE use combinations of cheaper and slower tools to achieve low cost solutions. Similarly to Part 2, this shows that there are advantages in returning multiple tool sequences to a decision maker.

Table 8.12. Showing the unique tool sequences used in solutions by NSGA-II (N), Guided Elitism (GE), Weighted Objective NSGA-II (WO), Precedence (P) and R-NSGA-II (R) on their individual highest scoring run according to the hypervolume indicator on Part 4 with the 18-tool library of mixed tool types.

Tool Sequence	Algorithm
8.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 8.0x4.0(ballnose)	P
8.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose)	GE
20.0x3.0(toroidal) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 8.0x4.0(ballnose)	P
20.0x3.0(toroidal) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose) – 6.0x3.0(ballnose)	WO
20.0x3.0(toroidal) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose)	GE,P,R
16.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose)	GE,R
12.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 8.0x4.0(ballnose)	P
12.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose) – 6.0x3.0(ballnose)	WO
10.0x0.0(end mill) – 8.0x1.0(toroidal) – 6.0x0.5(toroidal) – 10.0x5.0(ballnose) – 6.0x3.0(ballnose)	WO

Figure 8.16. Empirical attainment functions for the five algorithms applied to Part 1 using an 18 tool library. The red dashed line shows the best solutions found over 1000 runs. The green line shows the surface attained in the top 25% of runs. The blue line shows the surface found 50% of the time. The black solid line shows the surface attained in the 25% worst cases and the black dashed line shows the worst solutions found. Only solutions with 1.0mm of stock and under are considered.

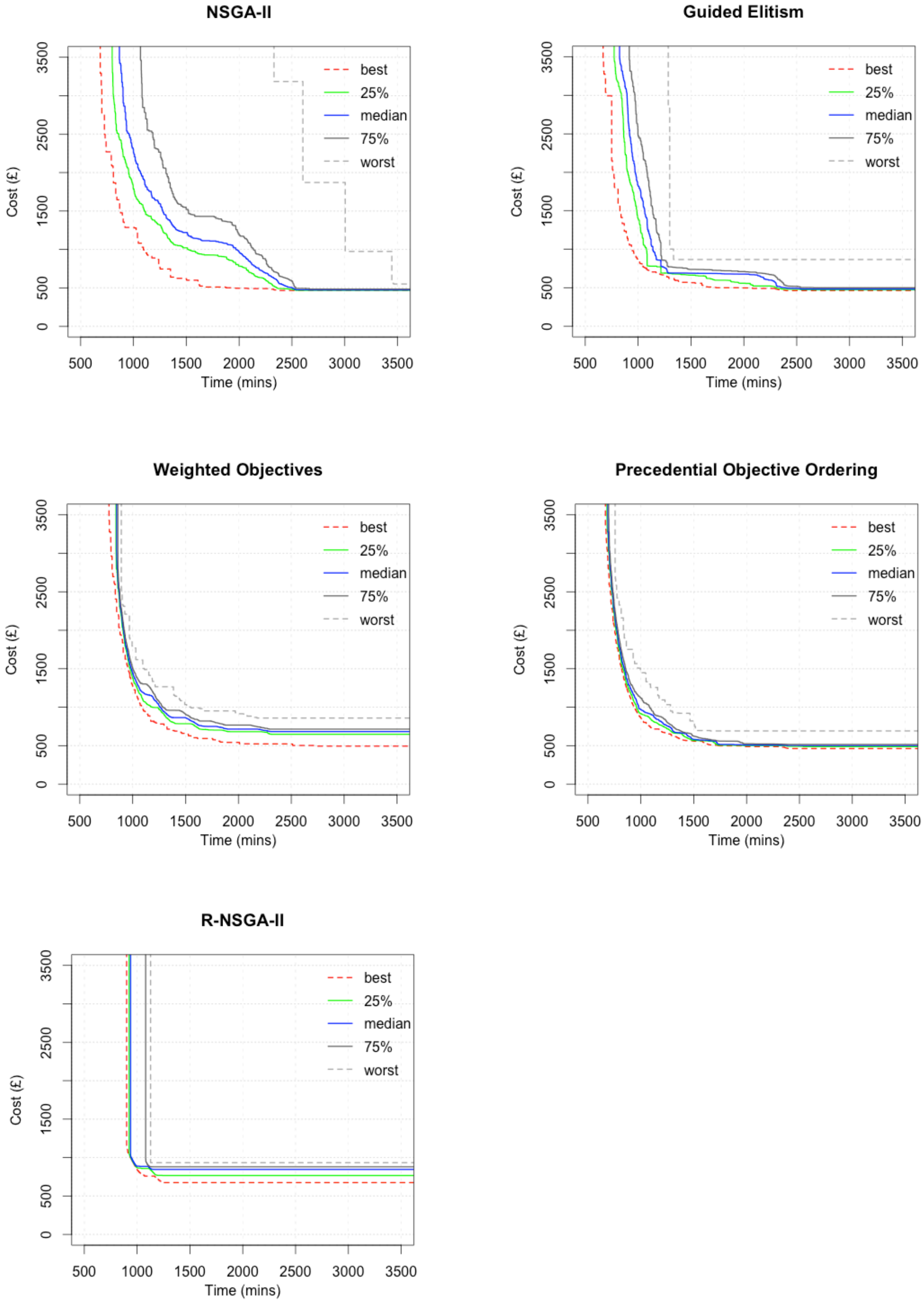


Figure 8.17. Showing for Part 4 with a library of 18 tools, the differences in Empirical Attainment Function between Precedential Objective Ordering and Guided Elitism; Precedential Objective Ordering and R-NSGA-II; Precedential Objective Ordering and Weighted Objectives; and Guided Elitism and NSGA-II. Coloured patches show the differences in attainment in favour of the algorithm named underneath the plot. White patches represent a small difference between 0% - 20% in frequency, while black patches represent large differences between 80% - 100%. On each graph, the median attained surface for the algorithm named underneath is shown with a dashed line. The best and worst surfaces found by the union of the sets returned by both algorithms are shown with solid lines.

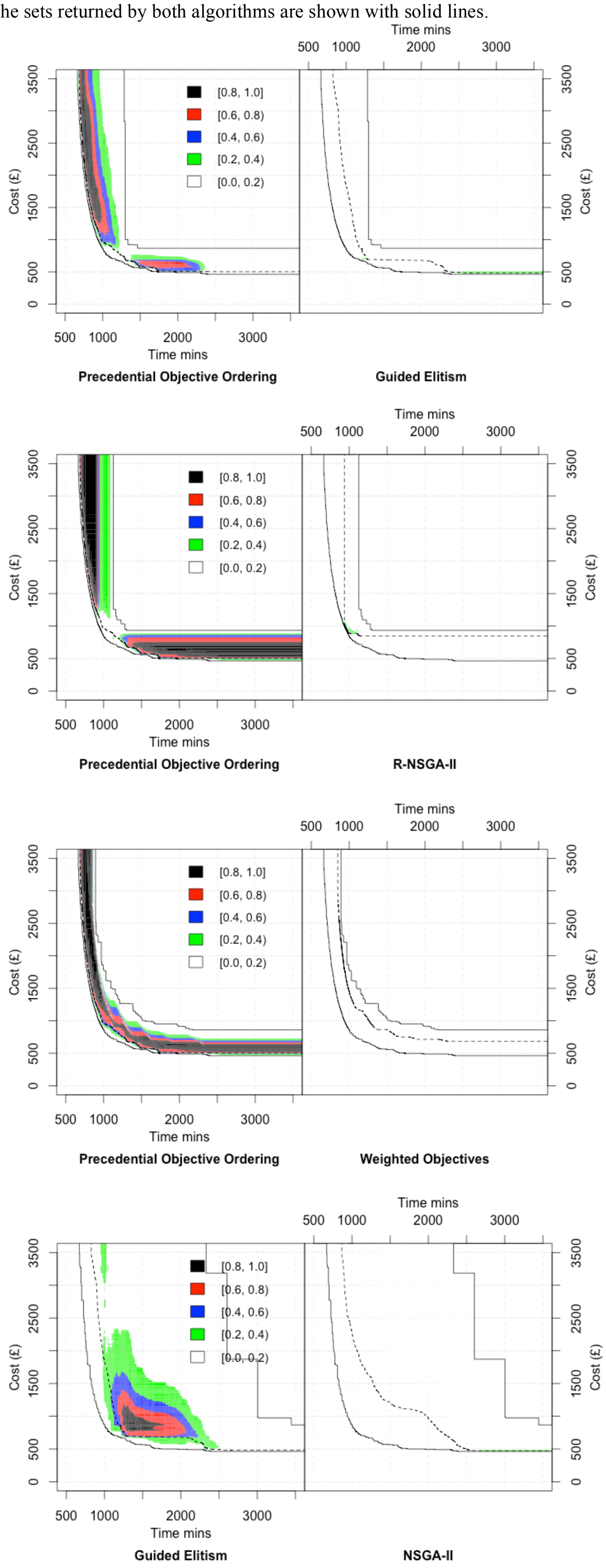
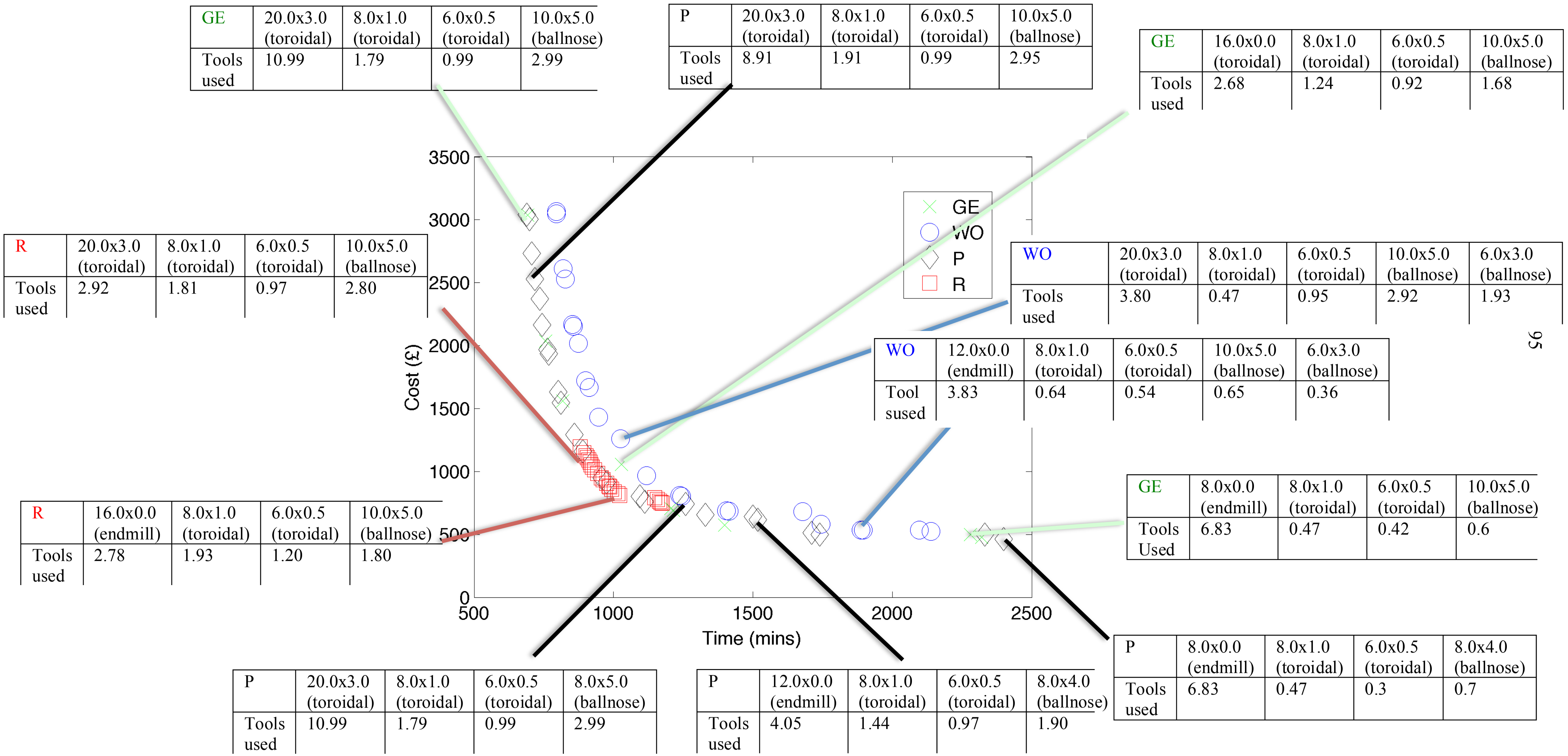


Figure 8.18. Showing the solutions found with excess material under 1mm for the highest scoring runs in terms of hypervolume for Precedential Objective Ordering (P), Guided Elitism (GE), R-NSGA-II (R), and Weighted Objectives (WO) on Part 2 with the 18 tool library of mixed tool types.



GE	20.0x3.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools used	10.99	1.79	0.99	2.99

P	20.0x3.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools used	8.91	1.91	0.99	2.95

GE	16.0x0.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools used	2.68	1.24	0.92	1.68

R	20.0x3.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools used	2.92	1.81	0.97	2.80

WO	20.0x3.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)	6.0x3.0 (ballnose)
Tools used	3.80	0.47	0.95	2.92	1.93

WO	12.0x0.0 (endmill)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)	6.0x3.0 (ballnose)
Tool sused	3.83	0.64	0.54	0.65	0.36

R	16.0x0.0 (endmill)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools used	2.78	1.93	1.20	1.80

GE	8.0x0.0 (endmill)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	10.0x5.0 (ballnose)
Tools Used	6.83	0.47	0.42	0.6

P	20.0x3.0 (toroidal)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	8.0x5.0 (ballnose)
Tools used	10.99	1.79	0.99	2.99

P	12.0x0.0 (endmill)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	8.0x4.0 (ballnose)
Tools used	4.05	1.44	0.97	1.90

P	8.0x0.0 (endmill)	8.0x1.0 (toroidal)	6.0x0.5 (toroidal)	8.0x4.0 (ballnose)
Tools used	6.83	0.47	0.3	0.7

8.5 Discussion

In this chapter we have, for the first time, produced a system that performs multi-objective tool sequence optimisation. The simultaneous optimisation of both tool sequences and cutting speeds creates a complex search space with both discrete and continuous properties. Nevertheless, the evolutionary multi-objective search algorithms have been shown to perform well. The completely unconstrained NSGA-II algorithm is able to find a diverse range of solutions within the required surface tolerance but methods with guidance and constraint handling unsurprisingly perform much better.

We performed experiments using real world tool life data from Machinery's Handbook (Oberg et al., 2008) on a small, simple search space and estimated Taylor Tool Life parameters on a much larger, complex search space. Broadly we achieved similar results, as in both experiments, the algorithms were able to find a diverse range of solutions offering a variety of trade-offs between total machining time and total tooling costs, while satisfying the constraint on excess stock. This shows that the presented system is feasible in a complex space and provides useful insights to decision makers.

The most interesting difference between the two experiments is in the tool sequences used. On the 12-tool experiment, only the smallest tool can reach the desired surface tolerance and all the algorithms converged on a two-tool sequence that was able to provide a good approximation of the Pareto front. In the 18-tool experiments, we see that there are many more tool sequence options and the best performing algorithms use a selection of tool sequences that specialise on certain regions of the Pareto front. For example, in the experiments on parts 1 and 4 we see smaller tools being used in solutions that find cheaper but slower results. Smaller tools are cheaper and so can provide for a low cost strategy, although this comes with longer machining times. Conversely, the fastest tool sequences often use larger tools but at a greater financial cost. This information shows the advantage of using a multi-objective approach, which has hitherto been ignored in Tool Sequence Optimisation. For example in (Ahmad et al., 2008) the optimisation uses a weighted aggregate function, combining machining time and manufacturing costs. This relies on a correct weighting being known a priori and does not provide different options to manufacturers.

In terms of algorithmic performance, the Precedential Objective Ordering method for constraint handling (P) consistently beats the other algorithms in terms of hypervolume. It has the best average score on all four tested search spaces. It finds a diverse range of solutions and is consistent across different runs. It is interesting that this method allows it to regularly locate the best solutions discovered by any algorithm and at the same time find a diverse range of solutions and tool sequences. The other constraint handling method, Weighted Objectives (WO), performs very well (essentially identically to P) on the simpler 12-tool search space but on the 18-tool experiments it performs relatively poorly. While it also returns a diverse range of solutions, in the experiments on parts 1 and 4 it appears to terminate in a suboptimal Pareto front, using different tool sequences to the other algorithms and finding dominated solutions. This implies that the use of the punishment factor is biasing solutions away from optimal regions. The technique used in P appears to allow a better and more unbiased exploration of the search space. However, in the 18-tool experiment on Part 2, P does not find as good solutions on the right hand side of the front as NSGA-II (N) and Guided Elitism (GE), which could suggest that it may be biased away from some regions of the front.

N and GE regularly find good solutions at the sides of the front. It may be that their constraint free approach allows them to better explore the search space. However, the algorithms do not converge as frequently as P or R-NSGA-II (R) on central regions of the Pareto front. The GE adaptation does help this in many cases, although it often converges onto a very narrow part of the central region of the front. This is especially true with the 18-tool experiment on Part 4, where there are fewer feasible solutions. On this experiment N was the worst performing of all the algorithms but GE returned consistently good solutions, showing that the guidance method helps concentrate on feasible solutions.

GE does not find a diverse range of solutions along the best discovered front in the same manner as P. Having said this, the adaptation can be considered successful because GE is the second best performing algorithm on all of the 18-tool experiments, finding a better range of solutions than N and regularly locating solutions that dominate WO. There are several potential ways that the performance of GE could be improved. The single-objective elitism could be turned off at a later stage to promote wider exploration nearer the end of search. There could also be a case for adding multiple weighted aggregate functions, which may be able to guide search in multiple directions. This would make the algorithm similar to MOEA/D (Zhang and Li, 2007).

In two of the experiments, R is the worst performing algorithm in terms of hypervolume, and is consistently outperformed by GE and P. However, this does not tell the whole story, as it consistently finds very good solutions in the central region of the front, which are as good as or better than those found by any of the other algorithms. However, it concentrates on a narrow selection of solutions that affects its hypervolume scores. This shows that it is a good method to use if decision makers have an idea of the particular region they want to find solutions in but not if they want to see the whole front. It is interesting to see that the same reference point can be used in all the experiments and locates a selection of central trade-off solutions. It may also be possible to improve performance by including extra reference points or switching between NSGA-II and R-NSGA-II's crowding distance methods at various points during the search process.

This work is one of the few that combines tool selection with machining parameter optimisation. (Spanoudakis et al., 2008) and (Wang and Jawahir, 2005) both restrict tool sequences to being exactly two tools in length and concentrate on a single-objective approach with only end mill tools. (Krimpenis and Vosniakos, 2008) takes a multi-objective approach but only uses single tool selection, not sequences of tools. The work in this chapter has used multiple tool types and shown that different regions of the Pareto front can be reached by using multiple tool sequences.

The algorithms in this chapter have worked in a two-stage manner. In the first stage, a combination of discrete Tool Sequence Optimisation and continuous Machining Parameter Optimisation takes place up to a set limit of discrete evaluations. In the second stage only continuous optimisation takes place. An extension of this work could be to take the feasible tool sequences discovered at the end of the first stage and then use specialised continuous methods such as MO-CMA-ES (Igel et al., 2007) to optimise the cutting speeds of the individual tool sequences, to produce a range of trade-off solutions for each tool. This could especially improve on the performance of N and GE, which continue to store infeasible solutions in their population during the continuous optimisation stage.

Particularly with the 18-tool experiment on Part 2, we see that the different algorithms find solutions in different regions of the Pareto front. One option to increase the diversity of tool sequences found and provide a better approximation of the front could be to use an ensemble of the presented algorithms. This could be achieved by using an island model with a different algorithm in each island, such as in (Leon et al., 2008). For example, P, GE and R could be on separate islands and migrate solutions between them.

8.6 Summary

This chapter has explored the first multi-objective system that optimises tool sequences with an associated machining parameter. Using a simpler tool library with realistic tool life information, a diverse range of solutions are found that offer interesting trade-offs between machining times and tooling costs. Using multi-objective search with constraint handling is shown to be particularly successful. In a second experiment, a more complex tool library is used, which is more relevant to real world machining but uses estimated tool life variables. The multi-objective algorithms are again shown to perform well, particularly the novel method for constraint handling, Precedential Objective Ordering, which is shown to be significantly more successful on complicated search spaces than using a punishment method. Preferential search through Guided Elitism and R-NSGA-II is also shown to consistently find high quality feasible solutions, although they do not find as good an approximation of the whole Pareto front as Precedential Objective Ordering. On all three parts tested on a larger search space with multiple tool types, the best performing runs of the algorithms find using multiple tool sequences enables them to cover different regions of the Pareto front. This suggests that the multi-objective approach to Tool Sequence Optimisation can provide very useful information to decision makers.

The work in this and previous chapters has been based around testing algorithms on a cached version of the search space, in order to evaluate the consistency of the stochastic based search techniques used by testing over a large number of trials. In the next chapter we apply parallel multi-objective algorithms to Tool Selection Optimisation, generating tool sequences and stock models in real time. This enables us to evaluate how feasible it is to use the evolutionary approach explored in this thesis in a real-world industrial setting.

Chapter 9

A Parallel Approach to Tool Sequence Optimisation

In this chapter we explore the use of parallel computing to speed up the time taken to optimise tool sequences using multi-objective Evolutionary Algorithms. Experiments are carried out on the multi-objectivised version of the Tool Selection Problem, investigated previously in Chapter 7. However, here, for the first time in this thesis, tool paths and stock models are generated using Computer Aided Manufacturing (CAM) software in real time, as they would be in real world implementations of these algorithms. The master/slave parallelisation approach is used to speed up search run time on an Internet connected network of computers that could be easily and inexpensively implemented on an existing office network or commercial cloud computing service. An asynchronous steady-state and a synchronous generational algorithm are compared in experiments on five components. The asynchronous algorithm is shown to provide significant speed up and provides a comparable approximation of the Pareto front, although the synchronous algorithm finds a better spread of solutions on some of the larger tested parts.

This chapter is organised in the following way. We first introduce the motivation behind the investigation. We then provide background on the master/slave model used for the parallel algorithms in this chapter as well as the other two main parallel approaches. Next we introduce the experimental conditions and parallel architecture used before describing the parallel algorithms employed in the investigation. Finally we present the results of experiments on five parts. Parts of this chapter appear in (Churchill et al., 2013d).

9.1 Introduction

Experiments throughout this thesis have used cached search spaces, taking advantage of the *Distributed Processing System* described in Chapter 3. This has enabled a thorough examination of the performances of different algorithms across a large number of trials. As Evolutionary Algorithms are stochastic, it is important to carry out this type of investigation to check their consistency. Operating with a low budget of evaluations has been a key focus of the development and tuning of these algorithms. In the real world, these evaluations are computationally expensive and optimisation has to be performed in a limited amount of time.

One advantage of population-based evolutionary algorithms is that they are naturally parallel. In this chapter we explore the use of parallel computing to speed up the time taken to optimise tool sequences using NSGA-II with the Guided Elitism adaptation, introduced in Chapter 6. The motivation here is to provide a blueprint for applying the evolutionary methods discussed throughout this thesis in a way that is both realistic in terms of computation time and affordable and accessible to workshops. The master/slave parallelisation approach is used to speed up search run time. The generation of tool sequences in CAM software produces heterogeneous evaluation times, as different tool sequences take differing amounts of time to generate. Here, we compare asynchronous steady-state search with a synchronous generational algorithm, to firstly assess the difference in run times between the two approaches and secondly to analyse whether there is a difference in the spread of solutions found. A new enlarged tool library is used, which increases the search space by a factor of 6 compared to the work in chapters 5, 6 and 7, and the algorithms are tested on five parts.

9.2 Background

In this section we describe the three main approaches that have been taken to parallelise Evolutionary Algorithms – Master/Slave, Island and Diffusion Models. The experiments in this chapter use the Master/Slave model but the other approaches are also discussed for completeness.

9.2.1 Parallel Evolutionary Search

Parallel Evolutionary Search refers to Evolutionary Algorithms that divide search among several different processors, working concurrently. Parallel search has two main advantages, improving the efficiency and effectiveness of search (Cantu-Paz, 2000; Coello Coello et al., 2007). By spreading computations among processors, more solutions can be evaluated at once, which can speed up the search process. This can make search more time efficient. As Genetic Algorithms use populations of solutions, they can be seen as being inherently parallel because individual solutions can always be evaluated concurrently. In terms of effectiveness, as we discussed in Chapter 2, one challenge for Genetic Algorithms is the premature convergence on sub-optimal solutions. One way this can be avoided is by maintaining diversity in the population. In certain parallel systems, populations operate separately but concurrently, making them less prone to converge to the same local optima. This can make parallel search more effective, meaning it can find better solutions (Whitely et al., 1999; Coello Coello et al., 2007). Genetic Algorithms have been parallelised using three main methods: *Master/Slave*, *Island* and *Diffusion*. These are discussed individually below.

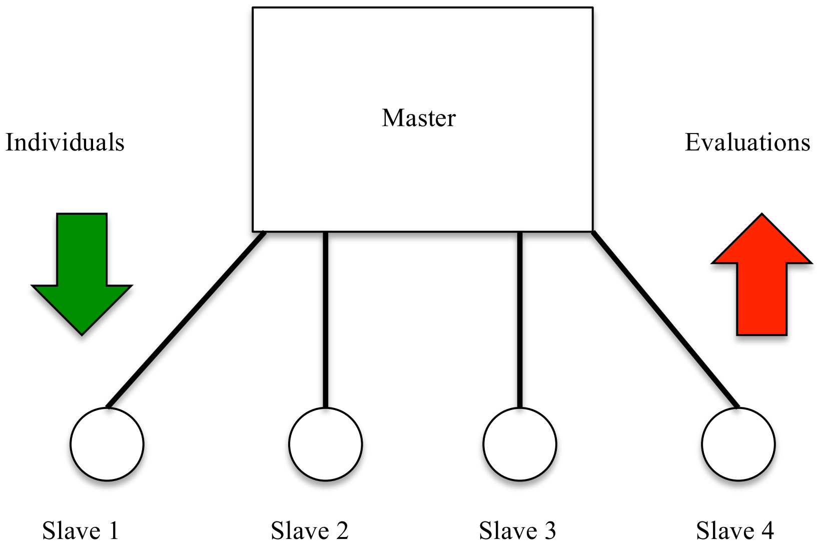
9.2.2 Master/Slave Model

The master/slave parallel model is the simplest of the three paradigms. In its basic form, there is a master processor, which controls the evolutionary algorithm. It sends individuals to slave processors, which perform a fitness function evaluation and return them to the master. In other variants, the master distributes subsets of individuals to slave processors or spreads the evaluation of a single individual among the slaves (Van Veldhuizen et al., 2003). An example of the master/slave approach can be seen in Figure 9.1.

The master/slave model is the most popular parallelisation scheme (Jaimes and Coello Coello, 2009). One reason for this is that it can be implemented without modifying the behaviour of the algorithm. In its basic form, it works exactly the same as a serial algorithm, with the computational cost of evaluations outsourced. The time taken can be estimated using the equation,

$T = P.T_c + \frac{n.f_i}{P}$ (9.1)

Figure 9.1. An example of a master/slave parallel system. The master sends individuals to the slave nodes who evaluate them and return their fitness values.



where P is the number of slave processors, T_c is the time taken to communicate with each slave, n is the size of the population and f_i is the time taken to evaluate the fitness of a single slave.

The simple master/slave model uses a synchronous generational strategy. A factor that reduces the efficiency of this approach is processors lying idle. From equation 9.1 we can see that if the communication costs are high compared to the evaluation costs there will be little speed up, and it could take longer than a serial implementation. Some individuals can take more time to be returned to the master than others, which is known as heterogeneous evaluation costs.

There are several causes of this, which are divided between issues of system architecture and problem specific function evaluations. In the first case, it may take longer for one processor to communicate with the master than another, due for example to being in different locations on an Internet connected system. The hardware of different slaves may also be heterogeneous. The speed of the processors themselves may be diverse, as well as the memory or file I/O conditions. In the second case, characteristics of individual solutions mean that some take longer to evaluate than others. For example, parameters encoded in a solution's decision vector may increase simulation times. Imagine a Knapsack Problem that uses a physical simulator to weigh items. The more items present, the more that need to be simulated, and hence a longer simulation time.

Durillo et al. compared synchronous and asynchronous approaches to parallelise NSGA-II through a Master/Slave approach (2008). They used three different algorithms: a standard synchronous generational NSGA-II (NSGA-II$_{\text{GEN}}$), an asynchronous generational NSGA-II (NSGA-II$_{\text{ASY-GEN}}$) and an asynchronous steady-state NSGA-II (NSGA-II$_{\text{SS}}$). NSGA-II$_{\text{ASY-GEN}}$ works by generating a new offspring as soon as the master receives a child from a slave. When the child population has been filled a new generation is created. This means that processors will have less idle time than with NSGA-II$_{\text{GEN}}$.

With NSGA-II$_{\text{SS}}$, the algorithm is adapted so that the auxiliary child population has a size of one. In the original algorithm the child population is the same size as the parent population, n, which means combined the parent and child populations have a size of $2n$. In the case of NSGA-II$_{\text{SS}}$, the combined parent and child population size is *n + 1*. A new generation event is triggered as soon as the master receives a child. If the child is fitter than any existing individual it replaces the weakest member of the previous population, as defined by Pareto ranking and crowding distance. At each generation event, as many children are produced as free processors and sent to the slaves for evaluation.

(Durillo et al., 2008) compared a serial version of NSGA-II$_{GEN}$ and NSGA-II$_{SS}$ on standard benchmarks, and found that for most test cases NSGA-II$_{SS}$ performed better. They suggest that this is due to the better precision offered by performing many more ranking and crowding events in the steady-state model. They also compare the three algorithms using a parallel structure on a real world application, optimising Mobile Ad hoc Networks (MANETs). They found that NSGA-II$_{GEN}$ and NSGA-II$_{SS}$ achieved similar quality results, and both were significantly better than NSGA-II$_{ASY-GEN}$. However, they found that NSGA-II$_{SS}$ was much faster than NSGA-II$_{GEN}$.

The real world experiment in (Durillo et al., 2008) had differing evaluation times due to heterogeneous hardware but the heterogeneity was not due to individual solutions. In (Yagoubi et al., 2011), a real world application is presented - diesel engine optimisation. In this problem, properties present in the solutions affect the evaluation times. Comparing NSGA-II$_{GEN}$ and NSGA-II$_{SS}$ using 40 processors for 1,600 evaluations, they found that NSGA-II$_{SS}$ performed better in terms of the Pareto front discovered and took much less time. On this problem NSGA-II$_{SS}$ did not seem to suffer from problems caused by solution-based heterogeneity. However, in (Yagoubi and Schoenauer, 2012) experiments on benchmark problems showed that if certain regions of the Pareto front are given much higher evaluation times, asynchronous steady-state algorithms do not cover this region as well as synchronous algorithms.

The Tool Selection Problem exhibits solution-based evaluation time heterogeneity, so it is important to evaluate differences in the results of both synchronous and asynchronous algorithms when using master/slave parallelisation.

9.2.3 Island Model

The Island model is also known as *multiple-deme*, *multiple population* or *coarse-grained parallelism* (Cantu-Paz, 2000; Coello Coello et al., 2007). In this paradigm, there are many subpopulations that are spread among different processors. These subpopulations evolve independently but at certain points individuals migrate from one island to another. In Figure 9.2 a two-island model is shown. Each island has its own population and there is migration between the two. Different migration policies and topologies can be used. For example, in systems with more than two processors, islands can be arranged in a ring, as seen in Figure 9.3. This is used in (Whitley et al., 1999), where each processor holds a separate subpopulation. After the first *x* generations, the top 5% of the population of each island replaces the least fit 5% of the island directly to the left, e.g. individuals migrate from island 1 to island 2. At the next *x* generations another migration takes place with islands two islands to the left, e.g. individuals migrate from island 1 to island 3. This process repeats, with the migrations moving around the ring.

Figure 9.2. An example of a two-island model with migration.

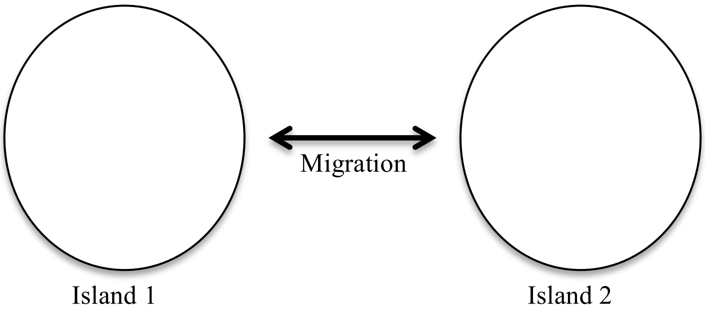
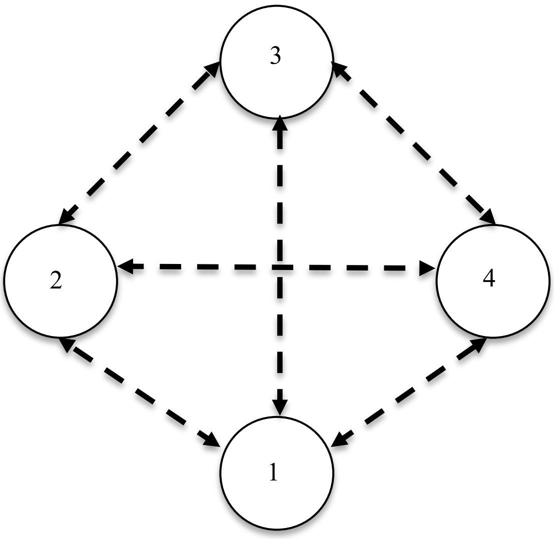


Figure 9.3. A ring island topology. Migrations can take place from one island to any other.



Island models allow several individuals to be evaluated in parallel and help maintain diversity. They also have the advantage of requiring limited communication between processors, which is a big concern for parallel algorithms. They have been applied to multi-objective algorithms in four main ways (Coello Coello et al., 2007; Talbi et al., 2008; Jaimes and Coello Coello, 2009). In the first method, all islands use the same multi-objective algorithms with the same parameters. In the second method, different islands use different parameters or multi-objective algorithms. For example, three different multi-objective algorithms are used in (Leon et al., 2008).

In the third method, each island evaluates different objective functions. For example, in (Okuda, 2002), there are m + 1 islands, where m is the number of objectives. One island applies a multi-objective algorithm to a subpopulation, while the other m islands implement a single-objective GA, evaluating one of the objectives each. The best solutions from the single-objective islands are migrated to the multi-objective island.

In the final method, each island, stored on an individual processor, is assigned to a different region of objective space. This can be difficult to achieve. One strategy is to keep generating individuals until enough have been found in the desired region (Van Veldhuizen et al., 2003). However, this can be computationally expensive. Another technique is to use preferential search to divide the islands into regions. For example, in (Deb et al., 2003) each island uses *Guided Dominance*, which was introduced in Chapter 5, with different weightings on the objective functions.

A rough estimate of the time taken to run the algorithm, T, is given by the equation,

$$T = G \, . \left(T_e + T_{mig} + T_{col}\right) \qquad (9.2)$$

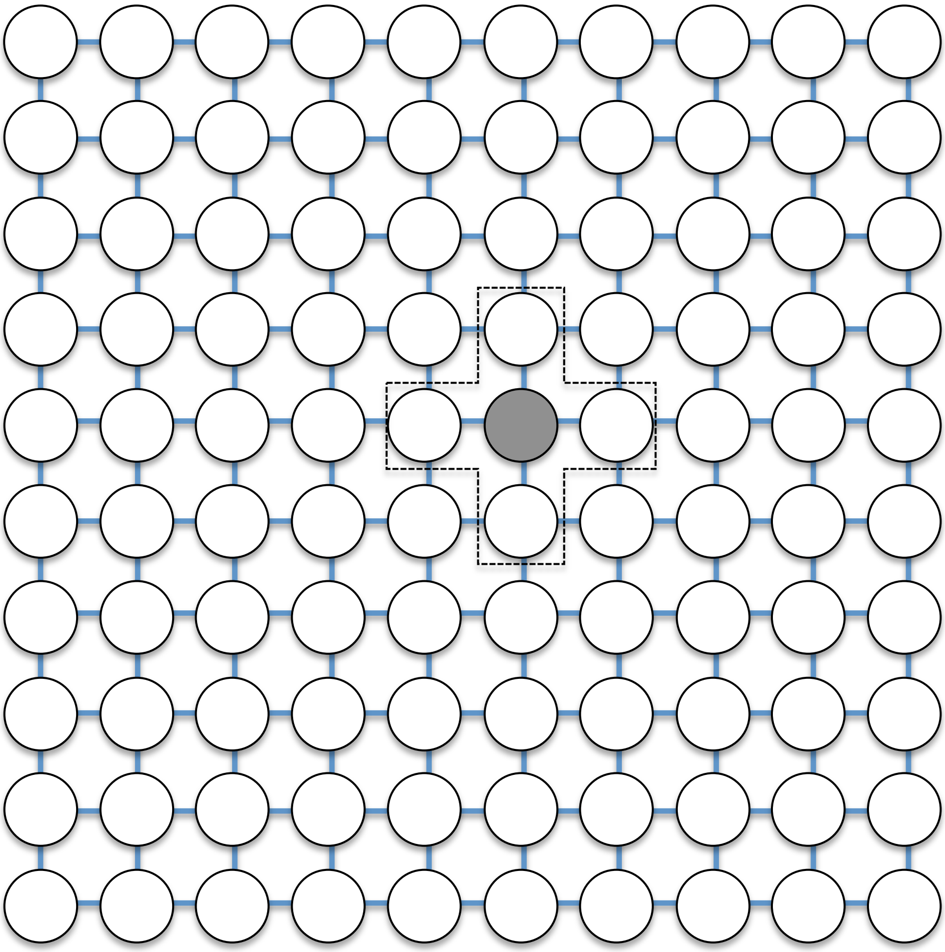
where G is the number of generations, T_e is the time taken to evaluate all members of the population on an island, T_{mig} is the time taken to migrate all individuals and T_{col} is the time taken to collect and operate on the population (Van Veldhuizen et al., 2003).

9.2.4 Diffusion Model

The diffusion model is also known as fine-grained parallelism. There is a single population that is connected in a specific topology. Each individual is only able to compete and reproduce with its neighbours, defined by a neighbourhood scheme. Typically each individual resides on a separate processor, although the algorithm can also be applied to serial systems (Cantu-Paz, 2000; Coello Coello et al., 2007). By having overlapping neighbourhoods, good solutions that develop locally can slowly diffuse around the population. This provides increased exploration by maintaining diversity in the global population, while allowing exploitation in the local neighbourhood (Nebro et al., 2009). Figure 9.4 shows an example of a diffusion model arranged in a 2d grid. The neighbourhood structure is a cross shape. There is a high communication cost between nodes in a neighbourhood, so the particular parallel computing platform being used will normally influence the choice of structure. The approximate expected runtime of the algorithm can be calculated using equation 9.2.

An example of a multi-objective diffusion model is MOCell (Nebro et al., 2009). The population is set up as in Figure 9.4. For each individual in the population, two parents are randomly chosen from the individual's local neighbourhood to create a child. The child is inserted into an archive, and also replaces the parent if it is better than it in terms of Pareto ranking and crowding distance. At the end of each generation, randomly chosen individuals from the archive replace individuals in the population.

Figure 9.4. An example of a diffusion model parallel system. Each circle is a node, which evaluates and stores an individual. The dotted cross shows the neighbours that the grey node can interact with.



9.3 Methods

In this section we will first discuss how the master/slave parallel approach has been applied to the Tool Selection Problem. We will then describe how time heterogeneity can occur in the evaluation of solutions. Next we will discuss the exact implementation of the parallel architecture used in the experiments below before introducing the two algorithms used, Synchronous Generational NSGA-II with Guided Elitism and Asynchronous Steady-State NSGA-II with Guided Elitism.

9.3.1 A Master/Slave Parallel Approach to the Tool Selection Problem

The master/slave paradigm is employed in the experiments in this chapter because it is easily applied to our existing system and allows us to spread out evaluations to reduce the number of generations or time steps. As discussed above, the asynchronous steady state algorithm was also shown to be efficient and effective in (Durillo et al., 2008). However, recent work by (Yagoubi et al., 2011) and (Yagoubi and Schoenauer, 2012) has shown that there could be problems in locating all regions of the Pareto front when faced with a problem that contains solution-based time heterogeneity. For this reason we compare the asynchronous steady-state algorithm with a synchronous generational one.

Up until now, the cost of evaluations has been carefully considered in the design and tuning of algorithms but the actual runtimes have not been explored. In the experiments below, asynchronous and synchronous algorithms are tested using five components with real-time generation of tool paths and stock models. To make it more applicable to real world situations and test the system on a larger scale problem, an expanded version of the tool library used in chapters 5, 7 and 8 is used, consisting of 26 tools. The parallelisation makes use of Internet connected heterogeneous computers, and could easily be implemented by a workshop over an office network or a cloud computing service.

9.3.2 Heterogeneous Evaluation Times in Tool Sequences

The synchronous and asynchronous algorithms are carefully compared below to assess whether the asynchronous algorithm is able to cover as diverse a range of solutions as the synchronous version. This is because there is solution-based evaluation time heterogeneity in tool sequences. Plans containing more tools take longer to simulate, as each tool must be processed sequentially. Tools with longer tool paths also take longer to calculate. For example, if both tools are able to remove all required material, when removing a cuboid from a rectangular blank, a 20mm diameter end mill will have a shorter tool path than a 2mm diameter end mill.

This time heterogeneity is further complicated by caching previously seen tool sequence fragments. When we generate the sequence:

Tool 1 – Tool 2 – Tool 3

we can store stock models and tool paths for:

Tool 1

Tool 1 – Tool 2

Tool 1 – Tool 2 – Tool 3

Any subsequent sequence that begins with those tools can proceed from the unseen part of the sequence. For example if we need to generate:

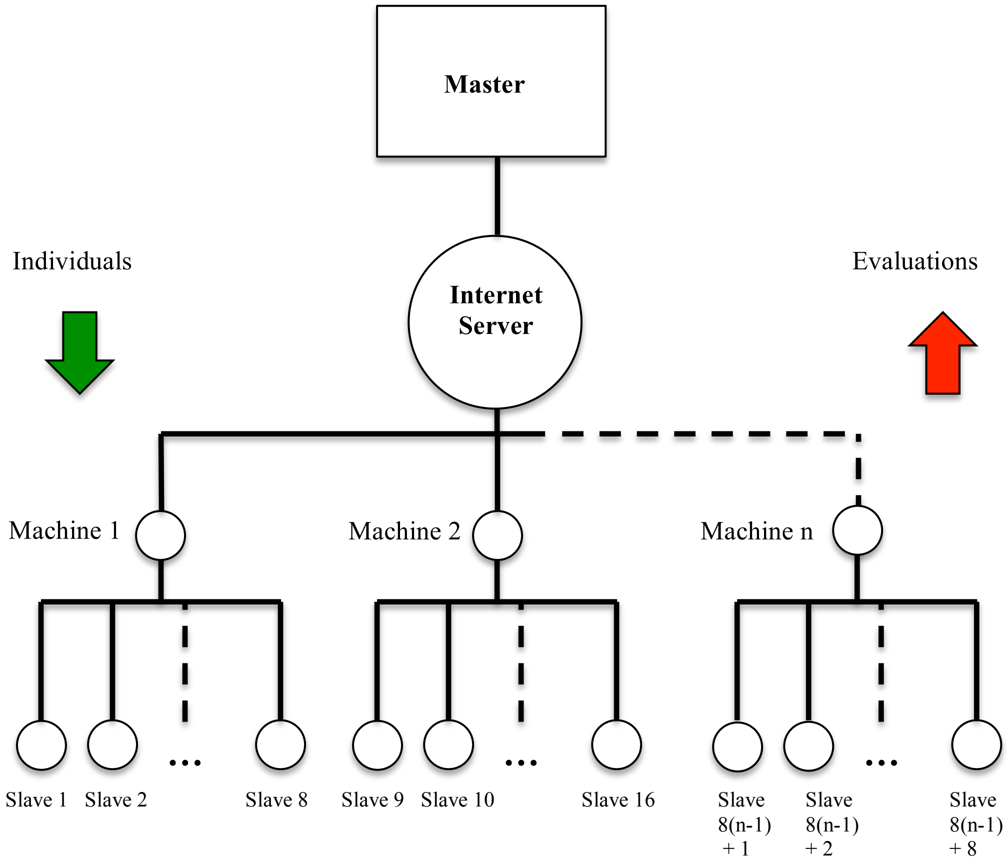
Tool 1 – Tool 2 – Tool 4

we can use the tool paths and stock models from the first two tools that were previously generated and only simulate the third tool. This means that depending on the current state of the cache, simulating a four-tool sequence could take less time than a three-tool sequence, or even a two-tool sequence.

9.3.3 Parallel Architecture

The parallel architecture used in the experiments below is shown in Figure 9.5. The system consists of one master processor, which controls the evolutionary algorithm, and either 3 or 5 slave machines. Each machine has an Intel Xeon four-core CPU @ 3.33ghz (model E31245). *Hyperthreading*

Figure 9.5. The parallel architecture used in the master/slave experiments. Each machine hosts 8 slave processors, which connect to the master over the Internet.



technology splits each core into two virtual cores, [9.1] and as such in the experiments below each machine is considered to have 8 processors. The slaves connect to the master via an Internet server.

Each slave simulates a tool sequence using *Machining Strategist 12* (Vero Software, 2012). This creates a tool path and stock model for each tool in the sequence, from which the objective function values are calculated and sent to the master. This creates a relatively large amount of data, as a stock model and tool path can be over 10mb. To avoid hefty data transfer, cached tool sequence fragments are only stored on an individual machine, and only accessible by the local slave processors.

9.3.4 Experimental Setup

9.3.4.1 Components and Tool Library

Simulated machining was carried out on five components, parts 1, 2, 3, 4 and 5, which are shown in Section 3.7 of Chapter 3. The work piece has been chosen as AISI 304 austenite steel. A tool library of 26 solid carbide coated tools was created from the ITC tool catalogue (ITC Ltd., 2012) and cutting parameters were chosen using the tooling manufacturer's guideline values. The library can be seen in Table 9.1, and contains 8 end mill, 8 ball nose and 10 toroidal tools.

[9.1] Information can be found at: http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html (accessed 18/1/2013)

As in the experiments in the previous chapters, tool sequences are bounded between 1 and 5 tools in length. Following the rules outlined in Chapter 3 to remove redundant sequences, using this tool library there are 218,341 attainable sequences in the experiments below.

9.3.5 Objective Functions

As in Chapter 6, two conflicting objective functions are considered, total machining time and the maximum thickness of excess stock:

$$f_1(S_p) = \ total\ machining\ time$$

$$f_2(S_p) = \ maximum\ thickness\ of\ excess\ stock$$

where *S* is a tool sequence applied to machining part, p. Again, the goal of search is to find solutions that achieve less than 1mm of excess stock in the fastest amount of time. In these experiments, the tools are assumed to be able to last for the entire machining process.

9.3.6 Algorithms

All of the experiments use the NSGA-II with the Guided Elitism adaptation (GE) that was introduced in Chapter 6 (see Section 6.7 for details). The representation scheme used is the same as that described in Chapter 5 (Section 5.3.1). Recombination also proceeds in the same fashion, while the "gradual mutation" method (also described in Section 5.3.1) is used for the mutation operator. The single-objective function used by Guided Elitism to protect individuals is:

$$f(\boldsymbol{E},\boldsymbol{p}) = \ T_E \ + \ k_E$$

$$c_E = \begin{cases} 5000, & d_E > 1.5mm \\ 1{,}000, & 1.0mm < d_E \leq 1.5mm \\ 0, & d_E < 1.0mm \end{cases} \qquad (9.3)$$

Table 9.1. Cutting parameters for the (left) end mill (middle) ball nose and (right) end mill tools that make up the tool library. d is the diameter of the cutter (mm), z is the number of teeth, C_r is the corner radius (mm), doc is the depth of cut (mm) and t_f is the linear table feed in mm/min.

End mill

d	z	C_r	doc	t_f
4	4	0	1.0	795.8
5	4	0	1.25	763.9
6	4	0	1.5	848.8
8	4	0	2.0	795.8
10	4	0	2.5	763.9
12	4	0	3.0	636.6
16	4	0	4.0	517.3
20	4	0	5.0	413.8

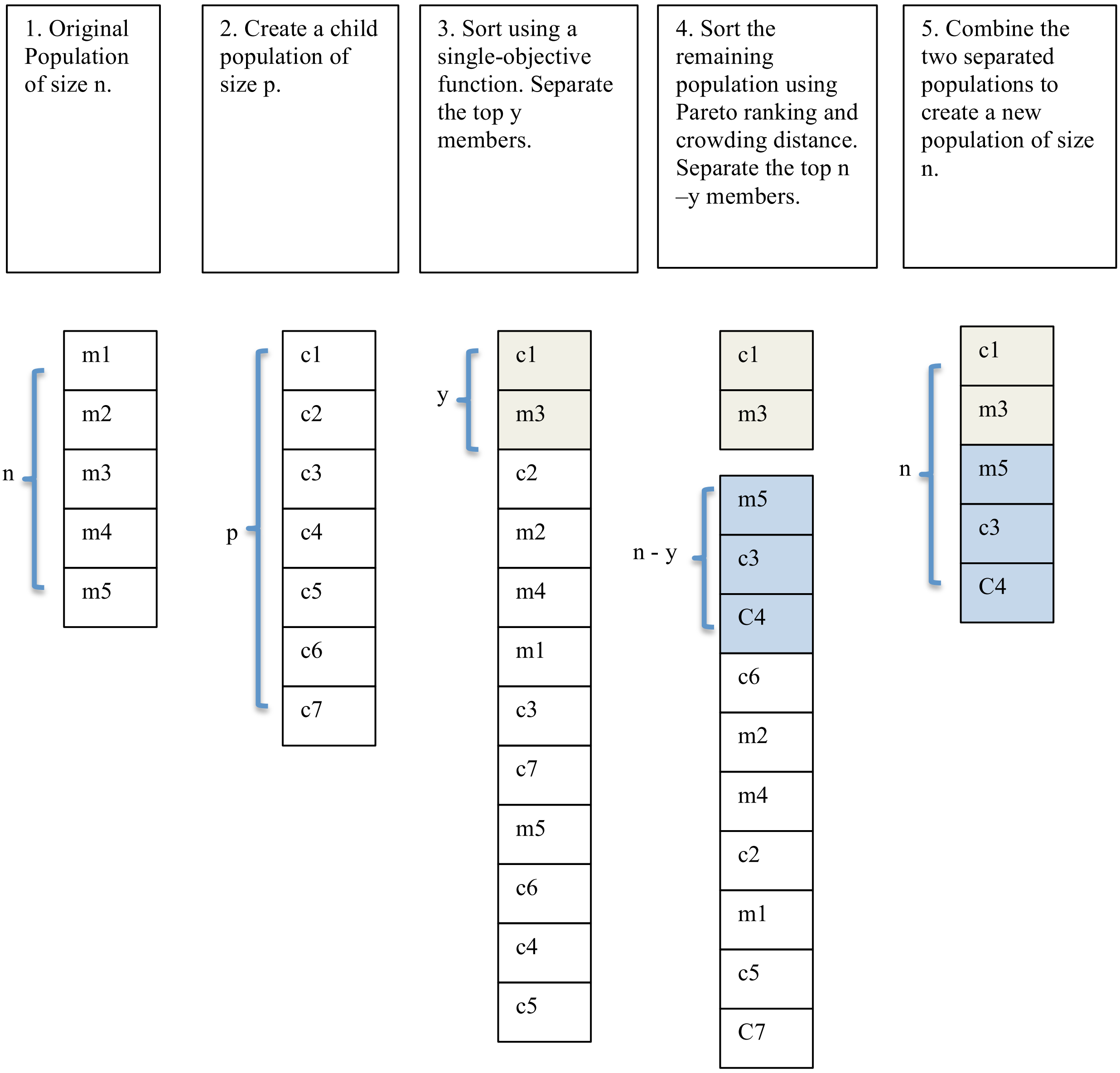
Ball Nose

d	z	C_r	doc	t_f
5	4	2.5	0.5	763.9
6	4	3.0	0.6	848.8
7	4	3.5	0.7	545.7
8	4	4.0	0.8	636.6
9	4	4.5	0.9	495.1
10	4	5.0	1.0	636.6
12	4	6.0	1.2	530.5
16	4	8.0	1.6	477.5

Toroidal

d	z	C_r	doc	t_f
5	4	0.5	0.5	763.9
6	4	0.5	0.5	848.8
8	4	1.0	1.0	636.6
8	4	1.0	1.0	636.6
10	4	0.5	1.0	636.6
10	4	1.0	1.0	636.6
12	4	1.0	2.0	530.5
12	4	1.5	2.0	530.5
12	4	2.0	2.0	530.5
16	4	2.0	2.0	477.5

Figure 9.6. An example of the synchronous NSGA-II algorithm with Guided Elitism. The population size is 5 and the number of processors is 7.



1. Original Population of size n.

2. Create a child population of size p.

3. Sort using a single-objective function. Separate the top y members.

4. Sort the remaining population using Pareto ranking and crowding distance. Separate the top n –y members.

5. Combine the two separated populations to create a new population of size n.

where T_x is the total machining time of tool sequence $\boldsymbol{x}$ and d_x is the maximum distance of excess stock.

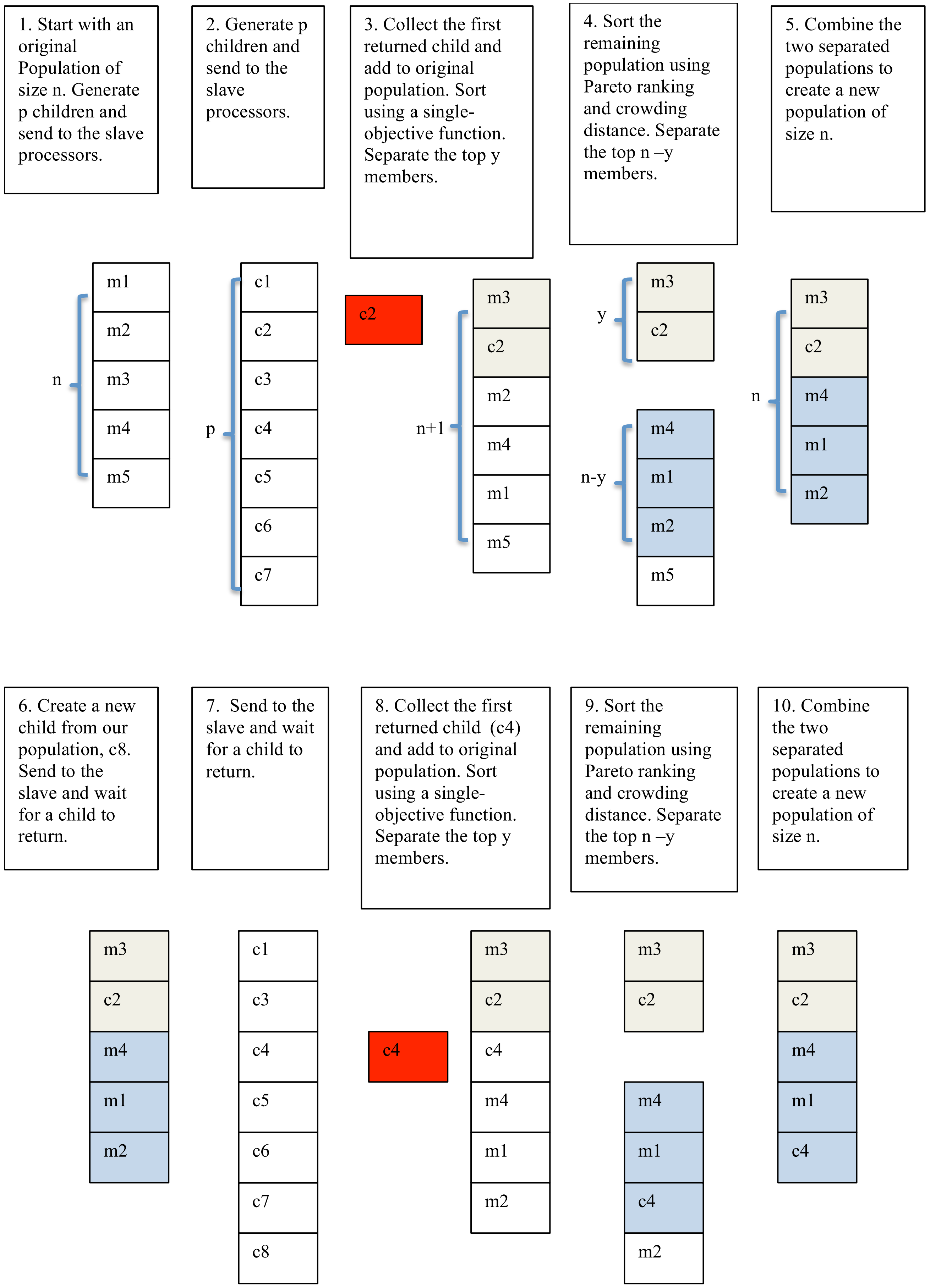
Two versions of GE are parallelised through the master/slave paradigm. The first, GE$_{SYN}$, uses synchronous generational evolution. In the case where the number of processors, p, is the same or smaller than the population size, n, at each generation n unique (previously unseen) offspring are produced. The offspring are then sent to the slave processors for evaluation. If p is greater than n, at each generation p unique offspring are produced and sent to the slave processors. Once all of the children have been evaluated and returned to the master, the child population is combined with the current population and then reduced to size p using Pareto dominance ranking and Crowding Distance. This means we use a $(\mu + \lambda)$ population generation scheme, where $\mu = n$ and $\lambda = p$.

Figure 9.6 shows an example of GE_{SYN}. Here, $n = 5$ and $p = 7$. We start with an initial population of 5. 7 children are then sent to the slave processors. When an evaluation of a child is complete it is sent back to the master. When all of the children have been returned to the master, the child population is combined with the initial population, creating a temporary population of size 12. This population is then sorted using a single-objective function and the top y members ($y = 2$) are separated. The rest of the population is sorted using Pareto ranking and Crowding Distance methods from NSGA-II and the top $n - y$ individuals, which is 3 in this case, are combined with the separated y individuals to create a new population of size n.

GE_{SS} uses asynchronous steady state evolution, as described in (Durillo et al., 2008). Here, a $(\mu + 1)$ population generation scheme is used, meaning that one child is created every generation. In order to parallelise the algorithm, x unique children are created and sent to a separate processor, where x is equal to the number of free processors. As soon as a child has been evaluated by the slave and received by the master, the population of size $n + 1$ is reduced to n using the normal Pareto dominance ranking and Crowding Distance methods found in NSGA-II, forming a new generation. The algorithm is asynchronous because children are not necessarily received by the master in the order that they are generated.

Figure 9.7 shows an example of GE_{SS}. Again, n = 5 and p = 7. As in Figure 9.6, we start with an initial population of 5. 7 children are then sent to the slave processors. When an evaluation is complete, the child is sent back to the master. As soon as a child is received, c2 in this example, it is combined with the initial population, creating a population of size 6. This population is sorted using a single-objective function and the top y members ($y = 2$) are separated. The rest of the population is sorted using Pareto ranking and Crowding Distance and the 3 individuals (n - y), are combined with the separated y individuals to create a new population of size n. The slaves are checked to see how many processors are idle. In this case only 1 is idle and so 1 child is generated and sent to this slave. The master waits for the next child to be returned, which is c4 in this example. The process repeats and a new population is generated.

Figure 9.7. An example of the asynchronous generational NSGA-II algorithm with Guided Elitism. The population size is 5 and the number of processors is 7.



1. Start with an original Population of size n. Generate p children and send to the slave processors.

2. Generate p children and send to the slave processors.

3. Collect the first returned child and add to original population. Sort using a single-objective function. Separate the top y members.

4. Sort the remaining population using Pareto ranking and crowding distance. Separate the top n –y members.

5. Combine the two separated populations to create a new population of size n.

6. Create a new child from our population, c8. Send to the slave and wait for a child to return.

7. Send to the slave and wait for a child to return.

8. Collect the first returned child (c4) and add to original population. Sort using a single-objective function. Separate the top y members.

9. Sort the remaining population using Pareto ranking and crowding distance. Separate the top n –y members.

10. Combine the two separated populations to create a new population of size n.

9.4 Results

In this section the synchronous generational and asynchronous steady-state algorithms are tested on five components to evaluate how they perform in terms of the total run time of the parallel algorithms, the best single-objective solution found and the range and diversity of solutions returned.

The results are divided between two sets of experiments. In the first section of experiments, Part 1 and Part 4 are tested with a system of 24 processors, run for 500 evaluations, and 40 processors, run for 1,000 evaluations. For each algorithm, 15 trials are performed using the 24-processor architecture and 15 trials are performed using the 40-processor architecture. In the second set of experiments, the two algorithms are tested on parts 2, 3 and 5 for 1,000 evaluations on the 40-processor system over 10 trials. Both algorithms use a population size of 24, a mutation rate of 0.7, a crossover rate of 0.6, and at each generation 2 solutions are protected through Guided Elitism.

When discussing the algorithms, GEN-P-X will refer to GE_{GEN} and SS-P-X will refer to GE_{SS}, where P is the number of processors and X is the evaluations limit.

9.4.1 Experiment 1

In this section we first analyse results for experiments on Part 1. We then analyse results for experiments on Part 4 before discussing the effect of increasing the number of processors but keeping the same population size.

9.4.1.1 Part 1

Table 9.2 shows the results for GE_{GEN} and GE_{SS} on Part 1. It is important to note that the results for 500 evaluations and 40 processors come from the same runs as 1,000 evaluations and 40 processors, i.e. they are snapshots of the 40 processor experiments from earlier in the evolutionary cycle.

The first thing that we notice is that all of the algorithms find a solution that takes 22.7 minutes, which we assume is the global optimum. This solution, {10.0x0.0(endmill) – 4.0x0.0(endmill)}, uses two tools, which makes it faster to evaluate than sequences containing more tools. From the boxplots in Figure 9.8, we see that GEN-40-500 converges on suboptimal solutions more frequently, which suggests that it may need more time (or generations) to converge. Figure 9.8 shows that SS-24-500 and SS-40-1000 both have less variability in hypervolume than the corresponding synchronous algorithms. The algorithms that run for 1,000 evaluations also achieve larger hypervolumes on average than at 500 evaluations.

Table 9.2. Showing the maximum (max), minimum (min), and median (mdn) total machining time of the best solution with excess stock < 1mm, total running time of the algorithm and hypervolume for GE_{GEN} and the GE_{SS} for different numbers of evaluations (Evals) and processors (Cpus) on Part 1. Superscripts on the median columns refer to significant differences (p <0.05, following the Bonferroni correction for multiple comparisions) with corresponding row according to a pairwise Wilcoxan Rank Sum test for equal distributions, if a Kruskal-Wallis test indicated that there was at least one significant difference. Hypervolumes are written in the form (h – 990,000), where h is the hypervolume obtained by the algorithms.

Experiment	**Machining Time (mins)**			**Running Time (mins)**			**Hypervolume**		
	Max	**Min**	**MDN**	**Max**	**Min**	**MDN**	**Max**	**Min**	**MDN**
NSGA-II_{GEN} 500 Evals 24 Cpus	22.7	22.7	22.7	27	22	$24^{2,4}$	5825.6	3780.4	5344.8^{5}
NSGA-II_{SS} 500 Evals 24 Cpus	22.7	22.7	22.7	15	13	$14^{1,5,6}$	5825.8	4253.9	5345.8
NSGA-II_{GEN} 500 Evals 40 Cpus	22.7	22.7	22.7	21	14	17^{5}	5826.9	3762.6	5171.6^{5}
NSGA-II_{SS} 500 Evals 40 Cpus	22.7	22.7	22.7	16	10	$12^{1,5,6}$	5824.4	3723.1	5237.4^{5}
NSGA-II_{GEN} 1,000 Evals 40 Cpus	22.7	22.7	22.7	37	29	$34^{2,3,4}$	5827.6	4254.4	$5619.8^{1,3,4}$
NSGA-II_{SS} 1,000 Evals 40 Cpus	22.7	22.7	22.7	27	21	$23^{2,4}$	5827.2	4962.2	5619.4

Figure 9.8. Boxplots showing the distributions of machining time (mins), running time (mins) and hypervolumes for the algorithms listed on the x-axis for experiments on Part 1, over 15 trials (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). GEN X P refers to GE_{GEN} and SS X P refers to GE_{SS}, where X is the evaluation limit and P is the number of processors. Hypervolumes are written in the form (h – 990,000), where h is the hypervolume obtained by the algorithms.

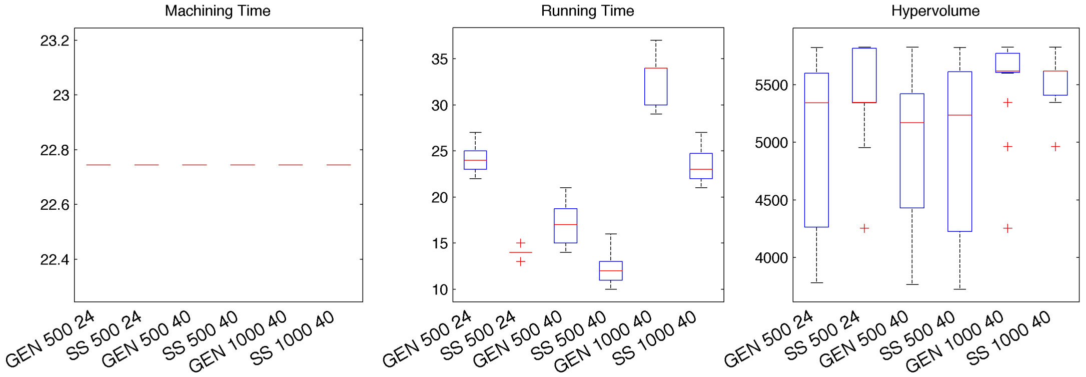


Figure 9.9, shows Empirical Attainment Function (EAF) plots for the GEN-40-1000 and SS-40-1000. We see that the algorithms perform comparably. They are both able to find a diverse range of solutions, particularly in the region that will be most interesting to a decision maker, the fastest solutions with excess stock under 1mm. The attained surfaces look alike for both algorithms, with similar Pareto fronts obtained in the best case. Figure 9.10 shows the differences between the two. This is negligible in most places but GE_{GEN} appears to find a region with longer machining time and lower excess stock more often, while GE_{GE} finds a small region with higher machining time more frequently.

In terms of running times, Figure 9.8 clearly shows that the asynchronous algorithms are superior. On average, SS-24-500 takes almost half the time needed by GEN-24-500. Using 40 more processors also results in a speed up for the synchronous algorithm. GEN-40-500 takes about 30% less time than GEN-24-500. Interestingly, using more processors does not greatly improve the speed of the asynchronous algorithm. SS-40-500 shows an average speed up of around 15% on SS-24-500, which implies that there could be a communication bottleneck.

Figure 9.10. Empirical attainment function plots for GE_{GEN} and GE_{SS} using 40 processors for 1,000 evaluations on Part 1. The top half of the figure shows the surface, while the bottom half shows the region with excess stock under 1mm.

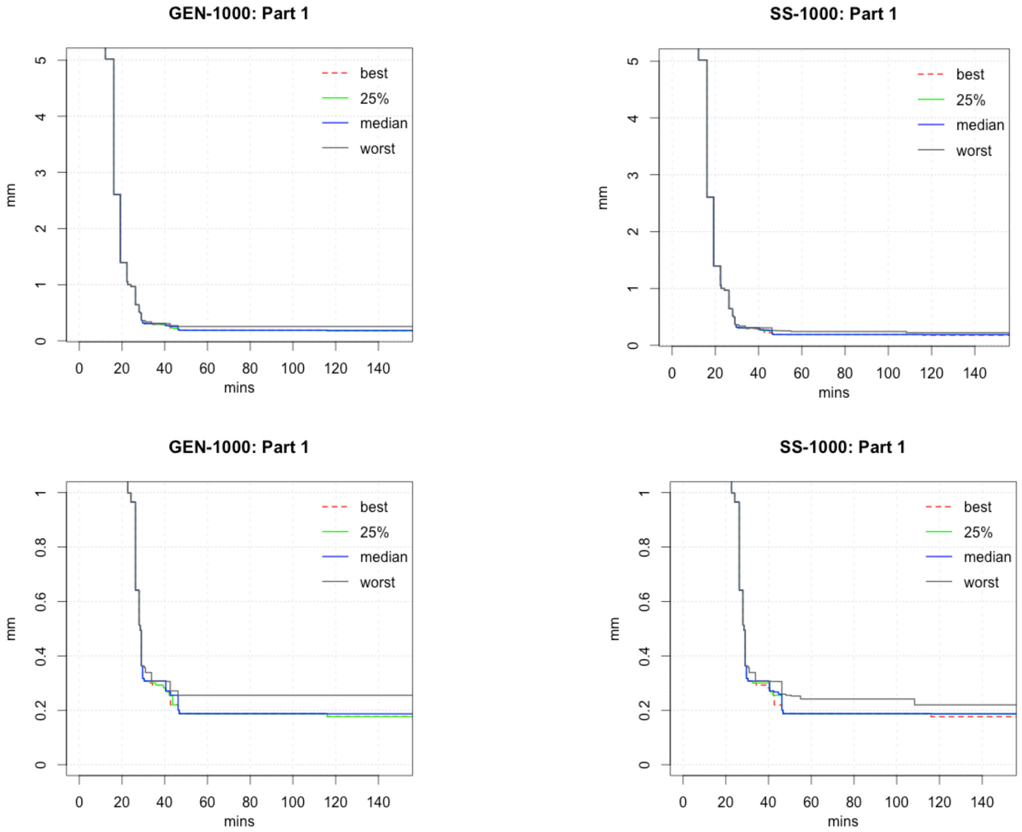
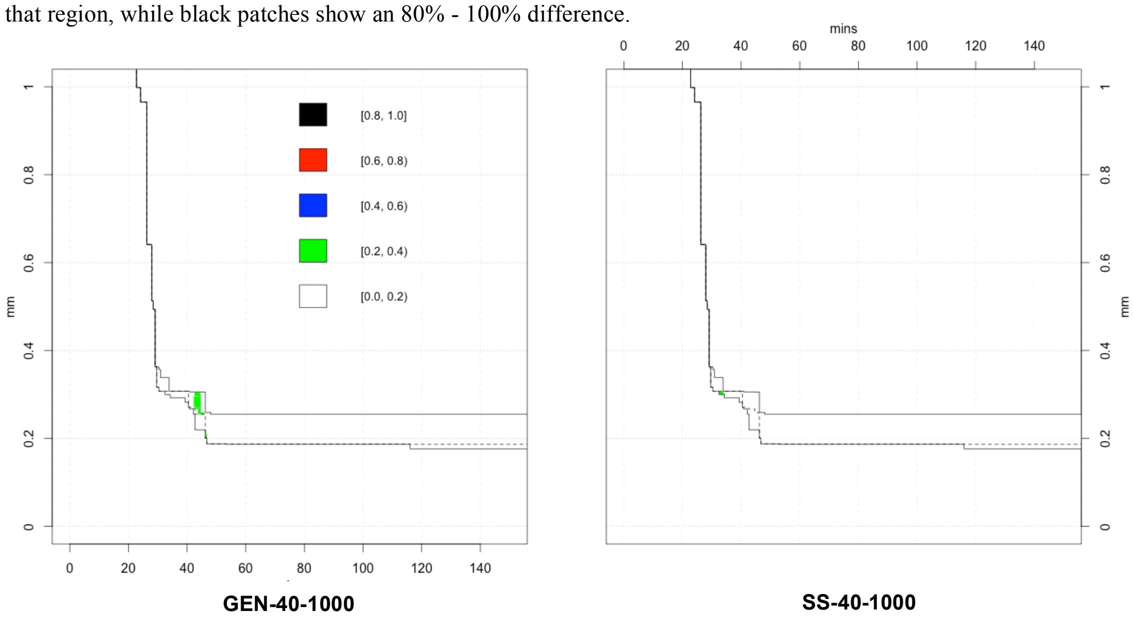


Figure 9.9. Plot showing the differences in empirical attainment function as a frequency interval between GE_{GEN} on the left and GE_{SS} on the right, on Part 1. Both use 40 processors and 1,000 evaluations. The left plot shows differences favouring GE_{GEN} and the right plot shows differences favouring GE_{SS}. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.



9.4.1.2 Part 4

The algorithms' performance on this part can be seen in Table 9.3 below. Part 4 is a larger component than Part 1 and we can see that both the machining time and the running time of the algorithms takes longer. The best solution from the single-objective perspective is,

{20.0x0.0(endmill) – 8.0x0.0(endmill) – 6.0x0.5(toroidal) – 5.0x0.0(endmill)}

Unlike with Part 1, none of the configurations are able to find this assumed single-objective optimal solution on every run. This solution consists of four tools, and so is likely to take longer to evaluate than shorter sequences. Both GEN-40-1000 and SS-40-1000 find this solution 13 times out of the 15 runs. At 500 evaluations, the algorithms using 24 processors perform better on average than the algorithms with 40 processors in terms of finding the single-objective optimal solution. This could be due to better exploitation. GEN-24-500 and SS-24-500 perform in an extremely similar manner, as do GEN-40-1000 and SS-40-1000.

In terms of hypervolume, GEN-40-1000 appears to perform better on average than SS-40-1000. This could suggest that the algorithm is suffering from problems related to heterogeneous evaluations. Figure 9.11 shows that on some runs GEN-40-500 and SS-40-500 attain higher hypervolume scores than GEN-24-500 and SS-24-500 can. This could indicate that using more processors can increase exploration, enhancing the breadth of solutions returned on this component.

Figure 9.12 shows EAF plots for GEN-40-1000 and SS-40-1000. They both find the Pareto optimal solutions (from the discovered front) with over 1mm of excess stock every time, and their performance is similar to each other with solutions under 1mm. We see that even when they do not obtain the assumed single-objective optimal solution, they get very close to it in the worst case. In the best case, the attained surfaces of the two algorithms are virtually indistinguishable from each other. However, the surfaces attained in the median and top 25% of runs are slightly better for GE_{GEN} in the region of lower excess stock and higher machining times. Figure 9.13 shows the differences in EAF for the two algorithms and we see that GE_{GEN} frequently performs better in this aforementioned region. This could point to issues with heterogeneous evaluation costs. Interestingly, GE_{SS} has slightly improved performance in some of the regions with lower machining times and higher excess stock.

Again, the asynchronous steady state algorithms have considerably shorter running times. SS-24-500 takes around 40% less time on average than GEN-24-500. SS-40-1000 displays similar savings on GEN-40-1000. Increasing the number of processors decreases running time for GEN-40-500 by around 30% compared to GEN-24-500. There is a much greater difference in the running times of the asynchronous algorithm that employs more processors than there is on Part 1. SS-40-500 shows an average speed up of around 30% on SS-24-500 as well, which is double the saving observed for the same pair on Part 1.

Table 9.3. Showing the maximum (max), minimum (min), and median (mdn) total machining time of the best solution with excess stock < 1mm, total running time of the algorithm and hypervolume for GE_{GEN} and the GE_{SS} for different numbers of evaluations (Evals) and processors (Cpus) on Part 4. Superscripts on the median columns refer to significant differences ($p < 0.05$, following the Bonferroni correction for multiple comparisions) with corresponding row according to a pairwise Wilcoxan Rank Sum test for equal distributions, if a Kruskal-Wallis test indicated that there was at least one significant difference. Hypervolumes are written in the form (h – 980,000), where h is the hypervolume obtained by the algorithms.

Experiment	**Machining Time (mins)**			**Running Time (mins)**			**Hypervolume**		
	Max	**Min**	**MDN**	**Max**	**Min**	**MDN**	**Max**	**Min**	**MDN**
NSGA-II_{GEN} 500 Evals 24 Cpus	306.7	300.4	304.9	98	77	$88^{2,3,4}$	1587.0	70.4	$1205.7^{5,6}$
NSGA-II_{SS} 500 Evals 24 Cpus	306.7	300.4	304.9	58	46	$54^{1,5,6}$	1463.9	760.4	$1288.2^{5,6}$
NSGA-II_{GEN} 500 Evals 40 Cpus	310.7	300.4	$305.7^{5,6}$	67	59	$64^{1,4,5}$	2125.0	1082.8	1427.3
NSGA-II_{SS} 500 Evals 40 Cpus	309.4	300.4	$305.7^{5,6}$	45	34	$39^{1,3,5,6}$	2154.7	737.7	1292.6
NSGA-II_{GEN} 1,000 Evals 40 Cpus	305.7	300.4	$300.4^{3,4}$	130	113	$120^{2,3,4,6}$	2270.7	1123.5	$1718.9^{1,2}$
NSGA-II_{SS} 1,000 Evals 40 Cpus	305.7	300.4	$300.4^{3,4}$	74	62	$70^{2,4,5}$	2292.5	1220.7	$1637.6^{1,2}$

Figure 9.11. Boxplots showing the distributions of machining time (mins), running time (mins) and hypervolumes for the algorithms listed on the x-axis on Part 4, over 15 trials (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). GEN X P refers to GE_{GEN} and SS X P refers to GE_{SS}, where X is the evaluation limit and P is the number of processors. Hypervolumes are written in the form (h – 980,000), where h is the hypervolume obtained by the algorithms.

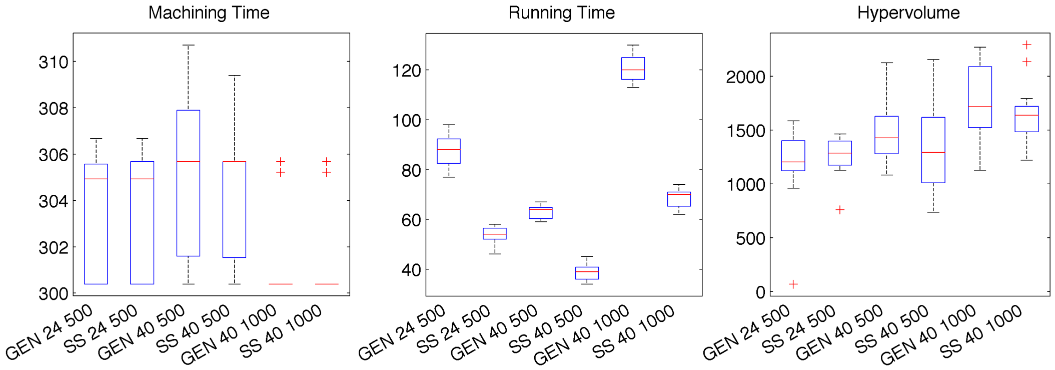


Figure 9.13. Empirical attainment function plots for GE_{SS} and GE_{GEN} using 40 processors for 1,000 evaluations on Part 4. The top half of the figure shows the surface, while the bottom half shows the region with excess stock under 1mm.

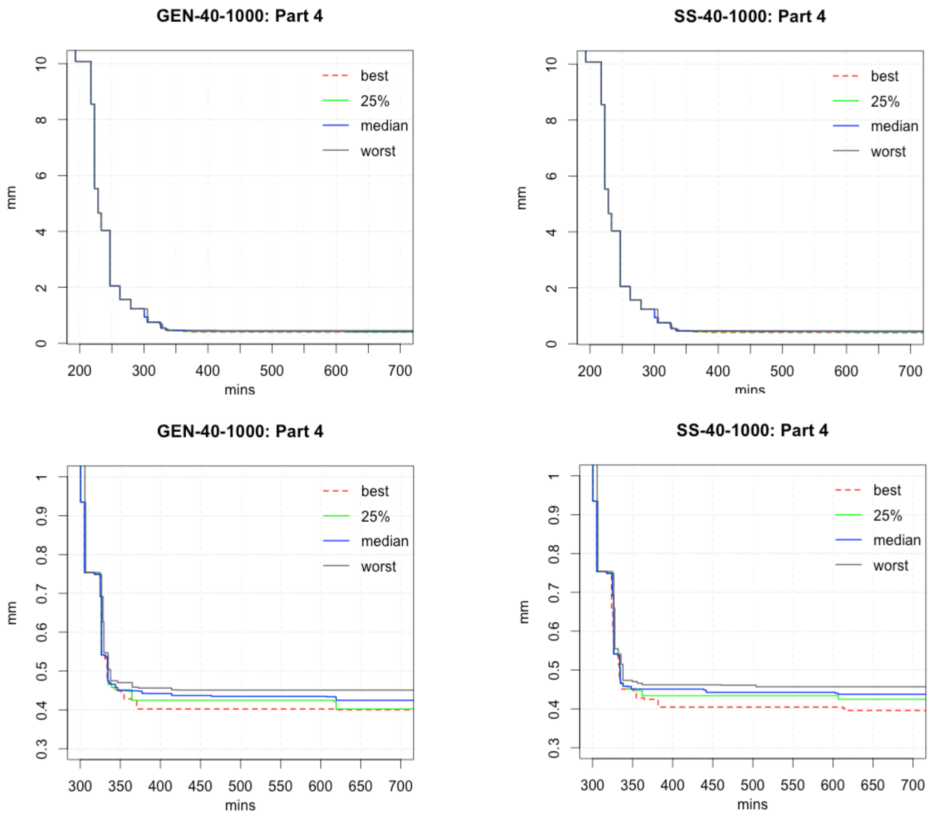
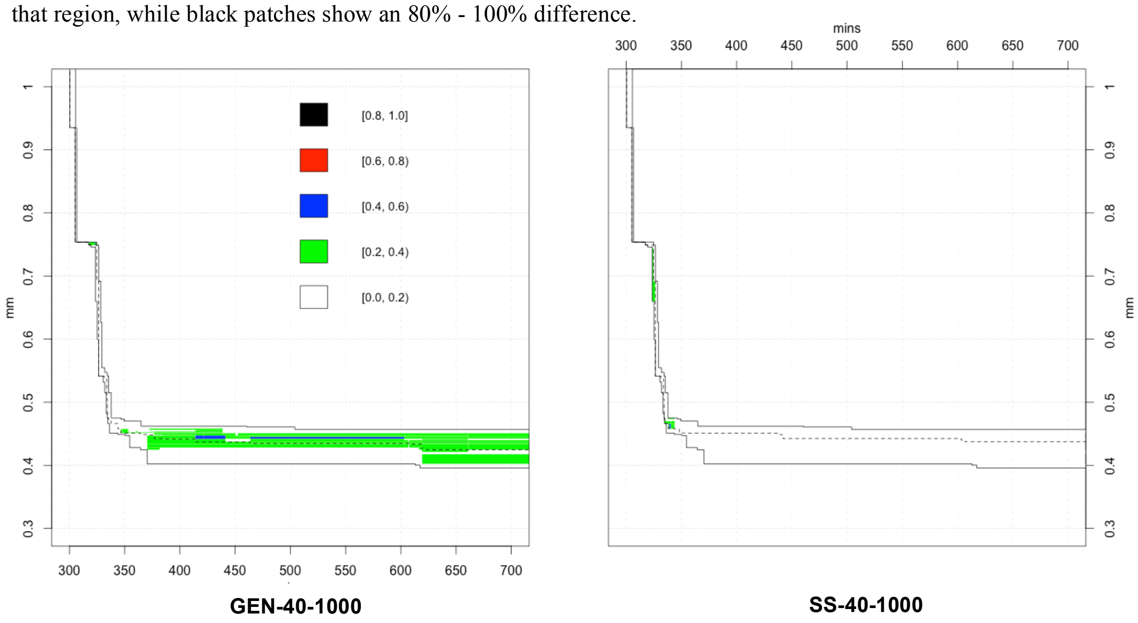


Figure 9.13. Plot showing the difference in empirical attainment functions as a frequency interval between GE_{GEN} on the left and GE_{SS} on the right, on Part 4. Both use 40 processors and 1,000 evaluations. The left plot shows differences favouring GE_{GEN} and the right plot shows differences favouring GE_{SS}. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.

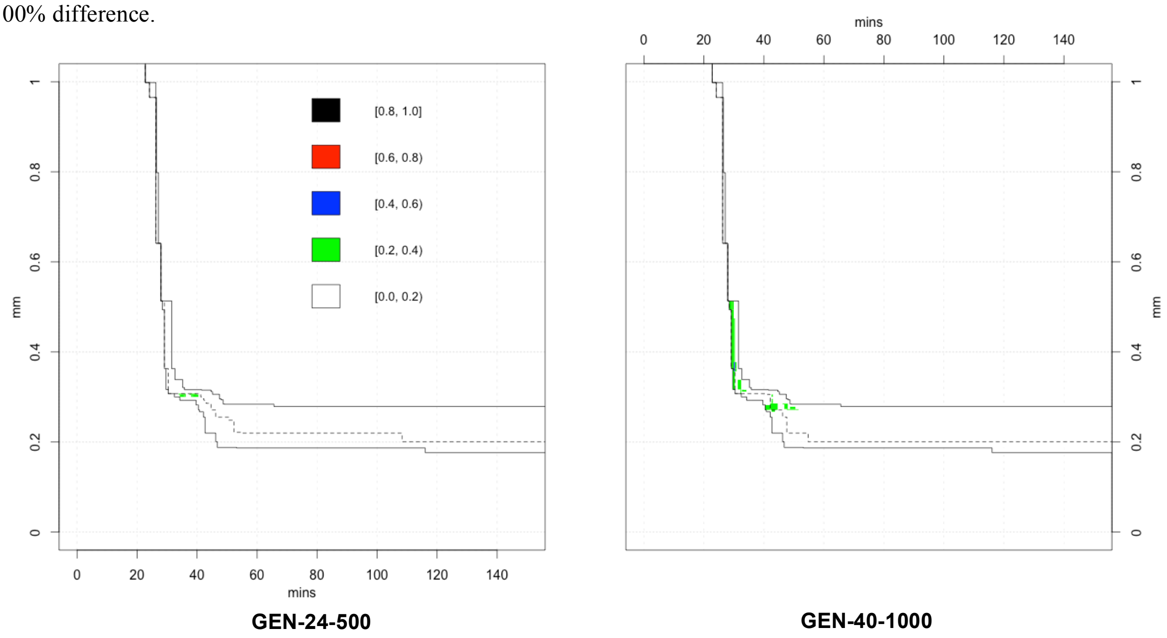


9.4.1.3 Using More Processors with the Same Population Size

All of the algorithms use the same population size but some use a different number of processors. This can change search behaviour. With GE_{GEN}, using 40 processors means that a child population of 40 individuals is generated, which is 16 more than the population size. Given the same evaluation budget, there will be fewer generations but more exploration at each generation. Figure 9.14 and Figure 9.16 show the difference between GE_{GEN} with 24 and 40 processors on parts 1 and 4 respectively, after 500 evaluations. On the first part, the two configurations of the generational algorithm perform similarly, with GEN-40-500 covering a slightly larger part of the front more frequently. On Part 4, the 40-processor algorithm more frequently attains solutions on the right side of the front, while the other achieves a better coverage of solutions with lower machining times. This suggests that using enlarged populations provides better exploration but running for more generations allows further exploitation.

The search behaviour of the asynchronous steady state algorithm is different from the generational one. Extra processors will evaluate children as soon as they are free. A new child will be generated from the current population. Given the same evaluation limit and population size, the version with more processors may evaluate a larger number of children that are related to solutions found earlier in the evolutionary cycle. In the experiments on Part 1, Figure 9.15 shows that at 500 evaluations, the algorithm that uses 24 processors frequently finds a considerably better spread of solutions. This is also reflected in the hypervolume scores. With Part 4, Figure 9.17 shows that the 40-processor configuration more often finds solutions on the right side of the front, while the other locates more solutions with a lower machining time, on the left side. As with GE_{SS}, this could suggest that differences in the number of processors running concurrently changes the exploitation/exploration balance.

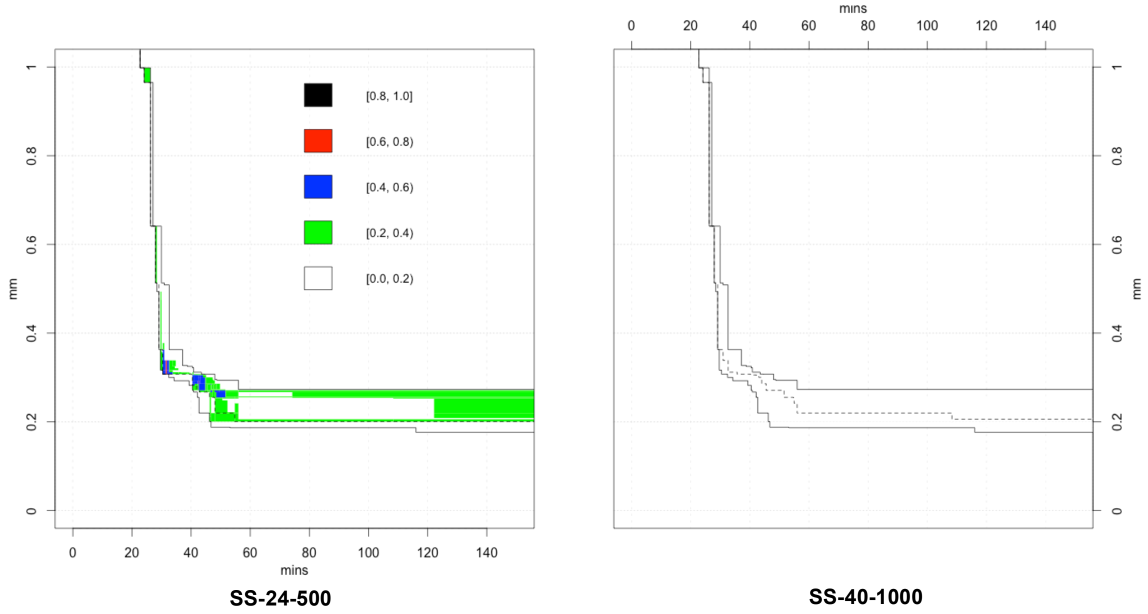
Figure 9.14. Plot showing for Part 1, the differences in empirical attainment functions as a frequency interval between GE_{GEN} using 24 processors on the left and 40 processors on the right, for 500 evaluations. Both use a population size of 24. The left plot shows differences favouring GE_{GEN} with 24 processors and the right plot shows differences favouring GE_{GEN} with 40 processors. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.



GEN-24-500

GEN-40-1000

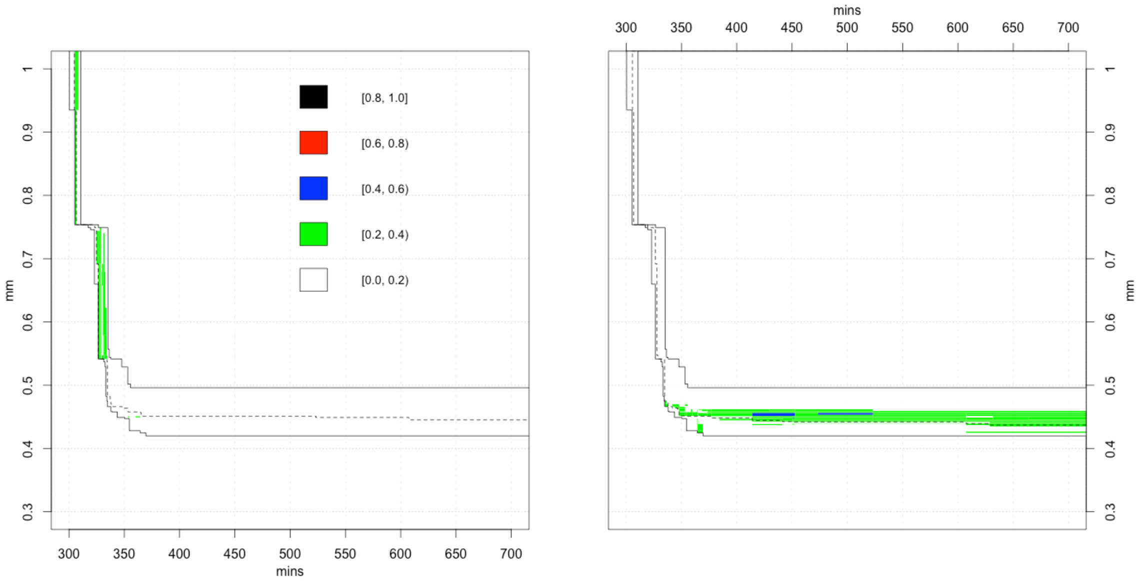
Figure 9.15. Plot showing for Part 1, the differences in empirical attainment functions as a frequency interval between GE_{SS} using 24 processors on the left and 40 processors on the right, for 500 evaluations. Both use a population size of 24. The left plot shows differences favouring GE_{SS} with 24 processors and the right plot shows differences favouring GE_{SS} with 40 processors. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.



SS-24-500

SS-40-1000

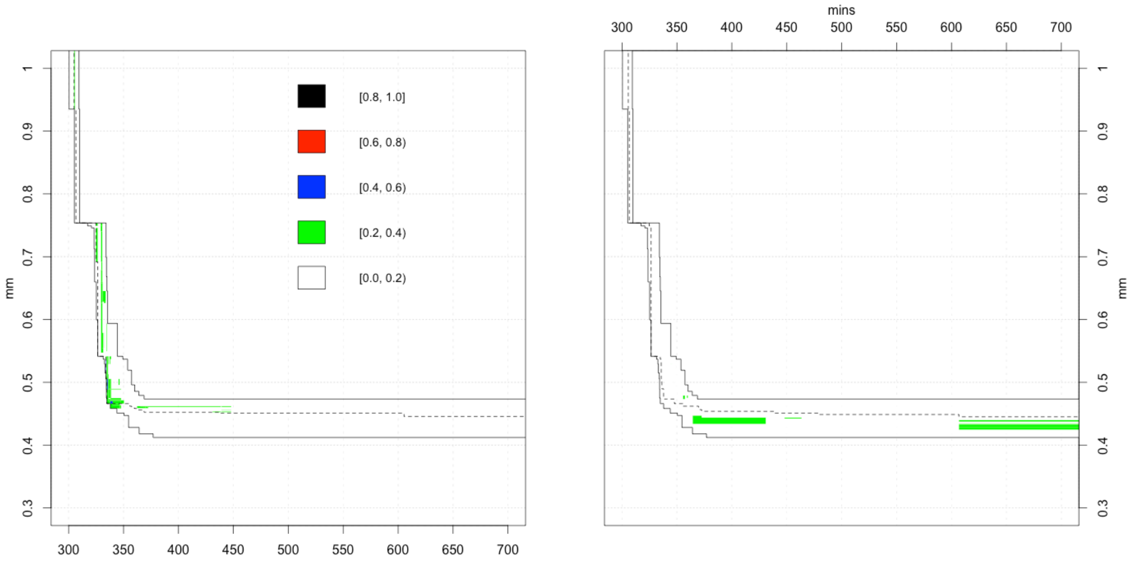
Figure 9.17. Plot showing for Part 4, the differences in empirical attainment functions as a frequency interval between GE_{GEN} using 24 processors on the left and 40 processors on the right, for 500 evaluations. Both use a population size of 24. The left plot shows differences favouring GE_{GEN} with 24 processors and the right plot shows differences favouring GE_{GEN} with 40 processors. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.



GEN-24-500

GEN-40-500

Figure 9.17. Plot showing for Part 4, the differences in empirical attainment functions as a frequency interval between GE_{SS} using 24 processors on the left and 24 processors on the right, for 500 evaluations. Both use a population size of 24. The left plot shows differences favouring GE_{SS} with 24 processors and the right plot shows differences favouring GE_{SS} with 40 processors. Surfaces are shown with under 1mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.



SS-24-500

SS-40-500

9.4.2 Experiment 2

In the second set of experiments, the 40-processor versions of GE$_{GEN}$ and GE$_{SS}$ are tested on parts 2, 3 and 5, over 10 trials each. Results are recorded after 500 and 1,000 evaluations.

9.4.2.1 Part 2

The fastest solution found on the trials on Part 2 with less than 1mm of excess stock contains two tools,

{10.0x0.0(endmill) – 8.0x1.0(toroidal)}

Table 9.4 shows that running search for 500 evaluations, neither of the algorithms are able to find this solution every time. For both algorithms, this is completely remedied by continuing search for another 500 evaluations. We can see in the boxplots in Figure 9.18 that at 1,000 evaluations the algorithms also have improved hypervolume scores, with less variability. On average, GEN-40-500 achieves better hypervolume than SS-40-500. After 1,000 evaluations, the algorithms achieve a very similar hypervolume score on average and look almost identical in the boxplot in Figure 9.18.

Although hypervolume scores are similar, Figure 9.19 shows that at 1,000 evaluations, there are differences in EAF between the algorithms. GE$_{GEN}$ appears to find solutions with the lowest excess stock more frequently (on the right side of the front), while GE$_{SS}$ seems better at locating solutions with higher excess stock and lower machining times. However, both algorithms are able to find a diverse range of solutions, even in the worse case. The asynchronous algorithm is again significantly faster than the synchronous one, taking around a third less time on average to perform 500 evaluations and 40% for 1,000 evaluations.

Table 9.4. Showing the maximum (max), minimum (min), and median (mdn) total machining time of the best solution with excess stock < 1mm, total running time of the algorithm and hypervolume for GE$_{GEN}$ and the GE$_{SS}$ for different numbers of evaluations (Evals) and processors (Cpus) on Part 2. Superscripts on the median columns refer to significant differences ($p < 0.05$, following the Bonferroni correction for multiple comparisions) with corresponding row according to a pairwise Wilcoxan Rank Sum test for equal distributions, if a Kruskal-Wallis test indicated that there was at least one significant difference. Hypervolumes are written in the form (h – 988,000), where h is the hypervolume obtained by the algorithms.

Experiment	**Machining Time (mins)**			**Running Time (mins)**			**Hypervolume**		
	Max	**Min**	**MDN**	**Max**	**Min**	**MDN**	**Max**	**Min**	**MDN**
NSGA-II$_{GEN}$ 500 Evals 40 Cpus	225.0	217.2	220.3^{3,4}	99	87	94.5^{3}	1411.3	630.9	1385.4^{3}
NSGA-II$_{SS}$ 500 Evals 40 Cpus	223.9	217.2	217.2	70	53	63.5^{3,4}	1387.1	917.3	1371.1^{3,4}
NSGA-II$_{GEN}$ 1,000 Evals 40 Cpus	217.2	217.2	217.2^{1}	200	173	187^{1,2,4}	1523.6	1387.6	1399.3^{1,2}
NSGA-II$_{SS}$ 1,000 Evals 40 Cpus	217.2	217.2	217.2^{1}	125	101	113.5^{2,3}	1526.8	1368.1	1399.9^{1}

Figure 9.19. Boxplots showing the distributions of machining times (mins), running times (mins) and hypervolumes for the algorithms listed on the x-axis on Part 2. GEN X P refers to GE_{GEN} and SS X P refers to GE_{SS}, where X is the evaluation limit and P is the number of processors (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). Hypervolumes are written in the form (h – 988,000), where h is the hypervolume obtained by the algorithms.

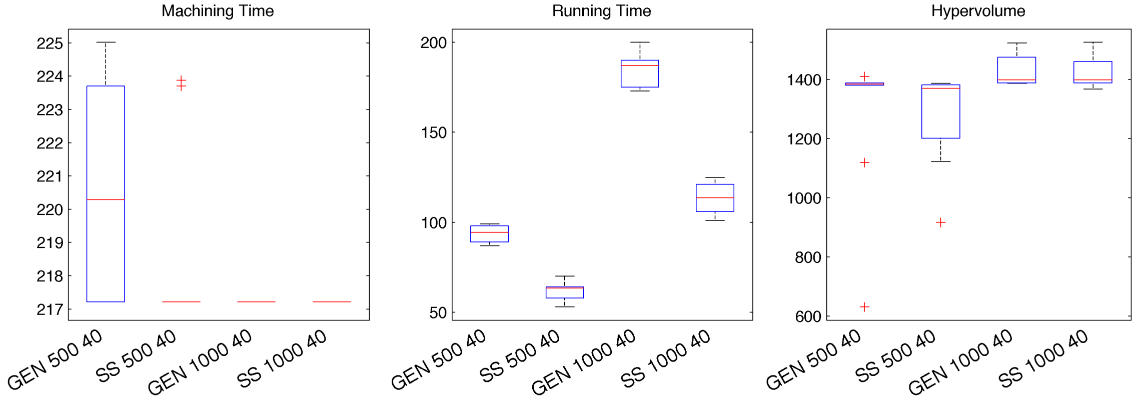
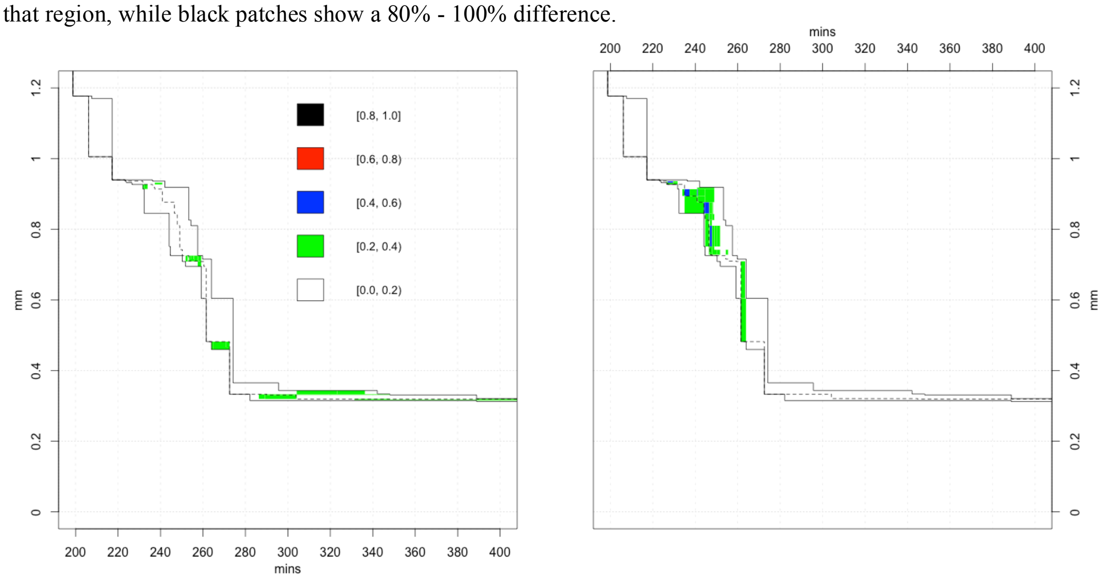


Figure 9.18. Plot showing the differences in empirical attainment function as a frequency interval between GE_{GEN} on the left and GE_{SS} on the right, on Part 2. Both use 40 processors and 1,000 evaluations. The left plot shows differences favouring GE_{GEN} and the right plot shows differences favouring GE_{SS}. Surfaces are shown with under 1.2mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show a 80% - 100% difference.



GEN-40-1000

SS-40-1000

Table 9.5. Showing the maximum (max), minimum (min), and median (mdn) total machining time of the best solution with excess stock < 1mm, total running time of the algorithm and hypervolume for GE$_{GEN}$ and the GE$_{SS}$ for different numbers of evaluations (Evals) and processors (Cpus) on Part 3. Superscripts on the median columns refer to significant differences (p <0.05, following the Bonferroni correction for multiple comparisions) with corresponding row according to a pairwise Wilcoxan Rank Sum test for equal medians, if a Kruskal-Wallis test indicated that there was at least one significant difference. Hypervolumes are written in the form (h – 990,000), where h is the hypervolume obtained by the algorithms.

Experiment	Machining Time (mins)			Running Time (mins)			Hypervolume		
	Max	Min	MDN	Max	Min	MDN	Max	Min	MDN
NSGA-II$_{GEN}$ 500 Evals 40 Cpus	51.3	41.3	41.3	31	26	28.5^{3}	4245.3	4236.0	4243.3
NSGA-II$_{SS}$ 500 Evals 40 Cpus	53.1	41.3	$42.4^{3,4}$	25	18	$19^{3,4}$	4245.7	4187.6	4242.6
NSGA-II$_{GEN}$ 1,000 Evals 40 Cpus	41.3	41.3	41.3^{2}	58	50	$54^{1,2,4}$	4247.1	4244.3	4246.8
NSGA-II$_{SS}$ 1,000 Evals 40 Cpus	41.3	41.3	41.3^{2}	39	33	$34.5^{2,3}$	4247.2	4245.7	4247.0

9.4.2.2 Part 3

The best single-objective solution for Part 3 uses a three tool sequence,

{10.0x0.0(endmill) – 8.0x0.5(toroidal) – 5.0x0.0(endmill)}

Again, as with Part 2, we can see in Table 9.5 that the algorithms are unable to locate this solution every time after 500 evaluations. GE$_{GEN}$ appears to perform slightly better at this evaluations limit. Both algorithms find this solution 100% of the time after 1,000 evaluations. As can be seen in the boxplots in Figure 9.20, the hypervolume scores between the algorithms are very close, which suggests that the Pareto front may be easier to approximate on this component than with Part 4, for example. This approximation is improved when the algorithms run for 1,000 evaluations. At this limit, the steady state algorithm slightly outperforms generational search in terms of hypervolume.

Looking at the differences in EAF in Figure 9.21, it seems to be the opposite situation to what we have seen on the other parts. GE$_{SS}$ more frequently attains solutions in the bottom right of the Pareto front, while GE$_{GEN}$ more frequently finds solutions in the region close to 1mm of excess stock. The generational algorithm finds the solution closest to the assumed single objective optimum, i.e. the second fastest solution found in all of the runs with excess stock less than 1mm, much more often. This solution is similar to the assumed optimal solution. The sequence is,

{12.0x0.0(endmill) – 8.0x1.0(toroidal) – 5.0x0.0(endmill)}

It is very close in objective space, as can be seen in Figure 9.21. It may be that the enlarged child populations enable the generational algorithm to better explore this area. For both algorithms, we see that the median attained surface is very close to the best. As with the other parts, GE$_{SS}$ is much faster than GE$_{GEN}$. It takes around a third less time over both 500 and 1,000 evaluations.

Figure 9.21. Boxplots showing the distributions of machining times (mins), running times (mins) and hypervolumes for the algorithms listed on the x-axis, labelled as described in x, tested on Part 3 processors (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). GEN X P refers to GE_{GEN} and SS X P refers to GE_{SS}, where X is the evaluation limit and P is the number of processors. Hypervolumes are written in the form (h – 990,000), where h is the hypervolume obtained by the algorithms.

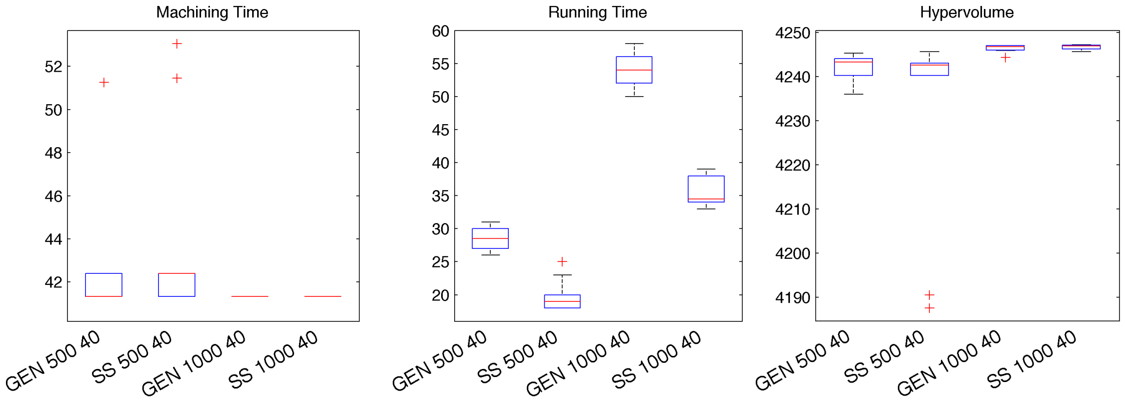


Figure 9.20. Plot showing the differences in empirical attainment function as a frequency interval between GE_{GEN} on the left and GE_{SS} on the right, on Part 3. Both use 40 processors and 1,000 evaluations. The left plot shows differences favouring GE_{GEN} and the right plot shows differences favouring GE_{SS}. Surfaces are shown with under 1.2mm of excess stock. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show a 80% - 100% difference.

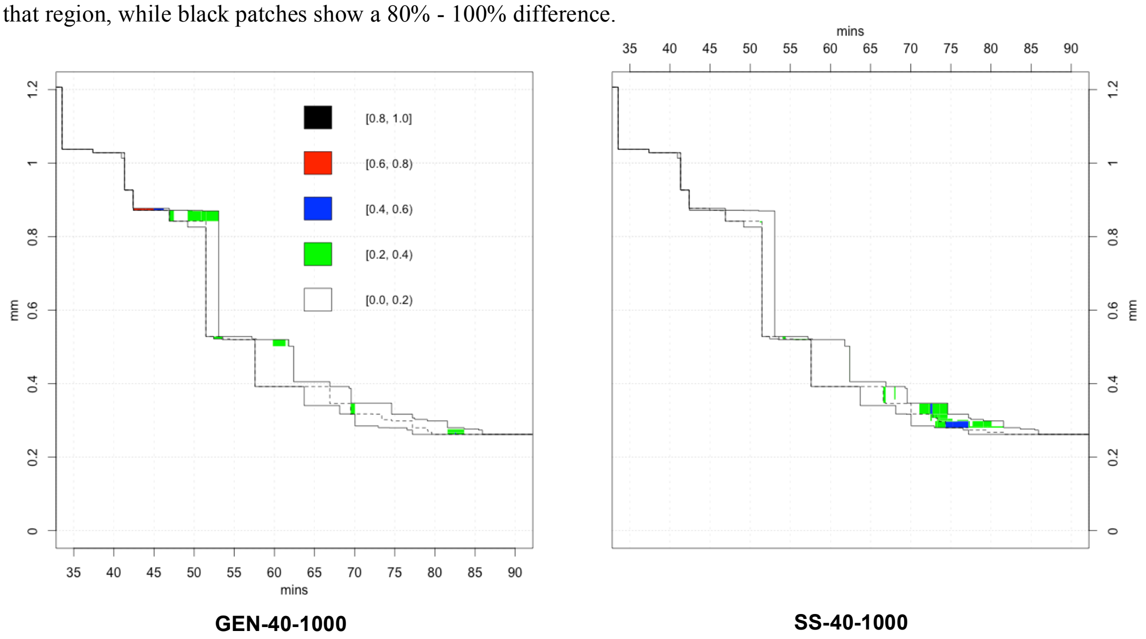
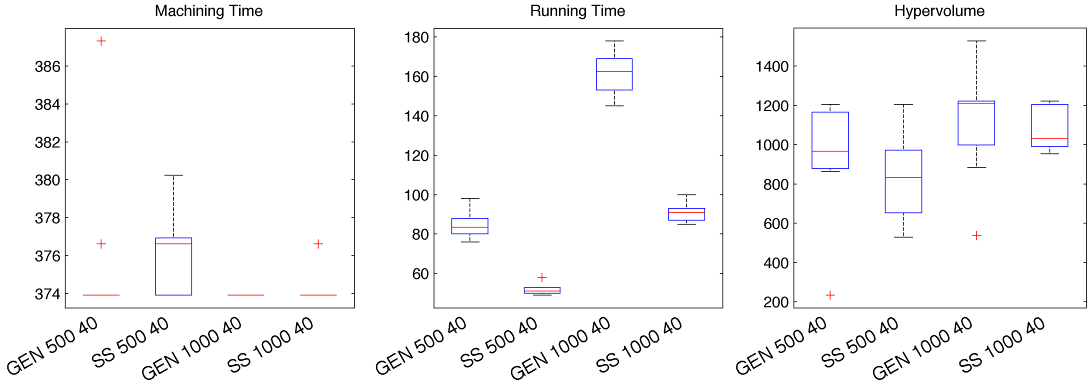


Table 9.6. Showing the maximum (max), minimum (min), and median (mdn) total machining time of the best solution with excess stock < 1mm, total running time of the algorithm and hypervolume for GE_{GEN} and the GE_{SS} for different numbers of evaluations (Evals) and processors (Cpus) on Part 5. Superscripts on the median columns refer to significant differences (p <0.05, following the Bonferroni correction for multiple comparisions) with corresponding row according to a pairwise Wilcoxan Rank Sum test for equal medians, if a Kruskal-Wallis test indicated that there was at least one significant difference. Hypervolumes are written in the form (h – 980,000), where h is the hypervolume obtained by the algorithms.

Experiment	**Machining Time (mins)**			**Running Time (mins)**			**Hypervolume**		
	Max	**Min**	**MDN**	**Max**	**Min**	**MDN**	**Max**	**Min**	**MDN**
NSGA-II_{GEN} 500 Evals 40 Cpus	387.3	373.9	373.9	98	76	83.5^{3}	1205.0	233.8	965.9
NSGA-II_{SS} 500 Evals 40 Cpus	380.2	373.9	$376.6^{3,4}$	58	49	$51^{3,4}$	1204.7	528.0	833.4^{3}
NSGA-II_{GEN} 1,000 Evals 40 Cpus	373.9	373.9	373.9^{2}	178	145	$162.5^{1,2,4}$	1528.4	537.5	1210.9^{2}
NSGA-II_{SS} 1,000 Evals 40 Cpus	376.6	373.9	373.9^{2}	100	85	$91^{2,3}$	1221.9	953.6	1033.2

Figure 9.22. Boxplots showing the machining time (mins), running time (mins) and hypervolumes for the algorithms listed on the x-axis on Part 5 (box: 25th and 75th percentile; central line: median; whiskers: extent of the data; crosses: outliers, defined as points that are beyond the quartiles by more than 1.5 times the inter-quartile range). GEN X P refers to GE_{GEN} and SS X P refers to GE_{SS}, where X is the evaluation limit and P is the number of processors. Hypervolumes are written in the form (h – 980,000), where h is the hypervolume obtained by the algorithms.



9.4.2.3 Part 5

As with Part 3, the fastest sequence found by the algorithms under 1mm for Part 5 uses a three tool sequence,

{20.0x0.0(endmill) – 16.0x2.0(toroidal) – 5.0x0.5(toroidal)}

In Table 9.6, we see that at 500 evaluations neither algorithm can find this solution on every run, although GE_{GEN} appears better able to do this than GE_{SS}. At 1,000 evaluations, the generational algorithm can locate this solution every time, while steady-state search finds this on all but one occasion. GE_{GEN} also appears able to achieve a better spread of solutions at both 500 and 1,000 evaluations. This is seen in the hypervolume scores, where the boxplots in Figure 9.20 show that the generational algorithm outperforms steady state search on average. GE_{GEN} appears considerably better in this regard at 500 evaluations, but the hypervolumes found in the top percentile of runs are similar for both algorithms at 1,000 evaluations.

The differences in EAF are shown in Figure 9.21 for the two algorithms after 1,000 evaluations, and are quite noticeable. Similarly to Part 2 and Part 4, the generational algorithm more frequently attains better solutions to the right of the Pareto front, with higher machining times and lower excess stock.

There is also a region between 0.6 and 0.8mm of excess stock that the generational algorithm locates more often. However, the asynchronous algorithm seems to better locate a region between 0.5 and 0.6mm of excess stock. The median performance, indicated with a dashed line is similar for both algorithms. The asynchronous algorithm is again considerably faster than the synchronous one, achieving around a 40% and 44% speed up at 500 and 1,000 evaluations respectively.

Figure 9.24. Plot showing the differences in empirical attainment function as a frequency interval between GE_{GEN} on the left and GE_{SS} on the right, on Part 5. Both use 40 processors and 1,000 evaluations. Surfaces are shown with under 1.2mm of excess stock. The left plot shows differences favouring GE_{GEN} and the right plot shows differences favouring GE_{SS}. White patches show that there is a 0% - 20% difference in frequency of solutions found in that region, while black patches show an 80% - 100% difference.

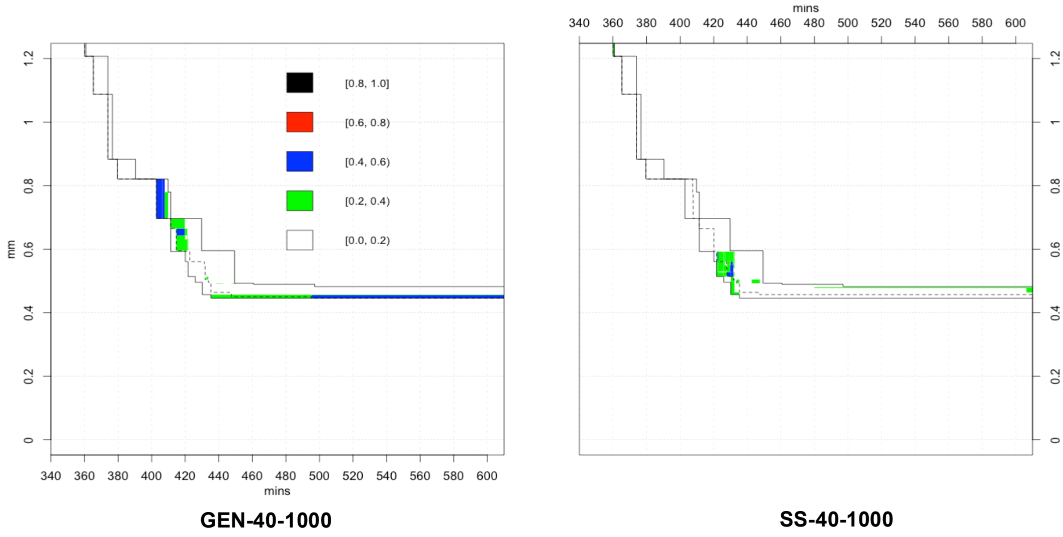
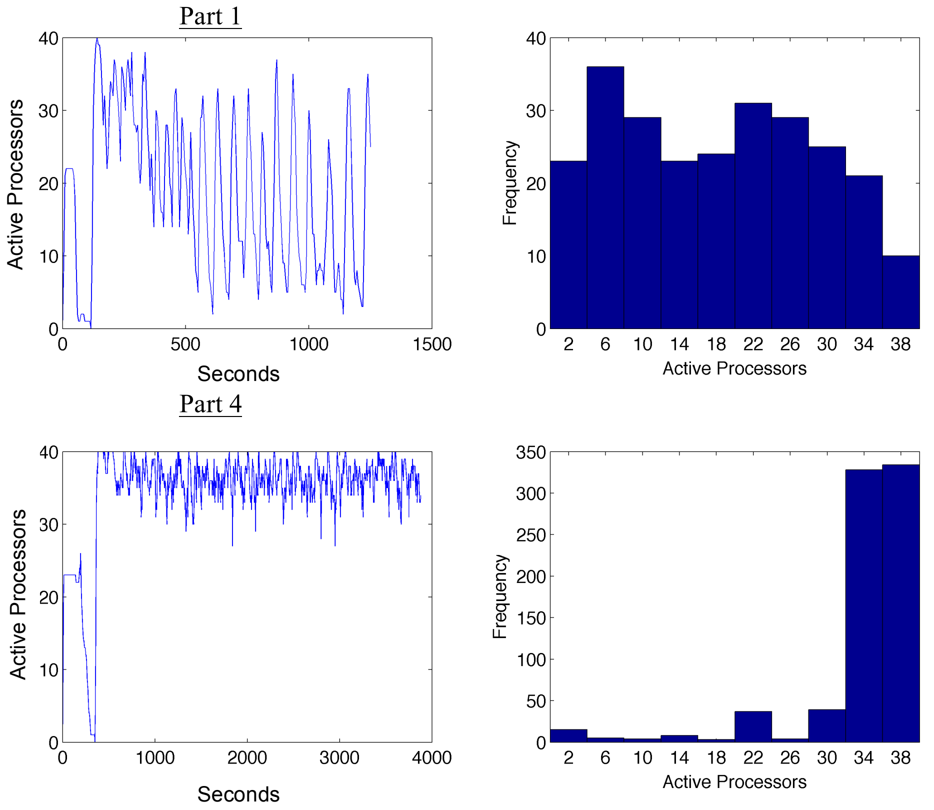


Figure 9.24. Plots showing the number of processors in use at 5 second intervals for GE_{SS} with 40 Processors on Part 1 and Part 4 when run for 1,000 evaluations.



9.5 Discussion

The experiments on the 5 parts have shown that NSGA-II with Guided Elitism is successful on this larger search space. Using 500 evaluations, both the synchronous and asynchronous algorithms are able to either reach the desired solution, i.e. the fastest solution found by any algorithm over all runs that met the surface tolerance requirement, or go very close to it on every run. They are also able to provide a diverse range of solutions. For both algorithms, this is improved by running for an extra 500 evaluations.

The master/slave parallelization approach is beneficial but not 100% efficient. In the first group of experiments, on parts 1 and 4, the algorithms were tested with 24 and 40 processors. Increasing the number processors, we would hope for a 40% decrease in runtime. GE_{GEN} sees slightly under a 30% decrease on both parts, while GE_{SS} achieves a 15% decrease on Part 1 and 30% on Part 4. The fact that we are not reaching the ideal time reduction suggests that there are factors that reduce the effect of parallelisation. This is particularly apparent for GE_{SS} on Part 1. These factors include the transfer times over the Internet and lag caused by the web server, file input and output costs, and the time it takes for the master to run each cycle of the evolutionary algorithm. For Part 1, it seems that we are approaching the threshold where extra processors can create speed gains. The evaluations on Part 4 take longer, so it is possible that using more processors could reduce speeds further. Figure 9.24 shows an example run for (a) Part 1 and (b) Part 4, using 40 processors and the asynchronous GE_{SS}. The number of active processors is recorded every 5 seconds. Here we can clearly see that processors are idle much more often on Part 1, which shows that the communication time affects the number of processors that can be used when the evaluation times are fast. Even on Part 4, which has much longer evaluation times, the system is rarely using all 40 processors. Avoiding the Internet server and communicating directly through the master could possibly help this problem. This could work well if there is a local cluster of computers available. However, the Internet connected system allows for the easy expansion of heterogeneous slave processors and handles failures well. Another method could be to assign multiple solutions to the slaves at once, so that they can process a new solution while they are communicating with the master. This could change search behaviour.

On both components, GE_{SS} has a considerably faster runtime than GE_{GEN}, when using the same number of processors and stopping condition. Across the parts, GE_{SS} generally takes 30% – 40% less time. This suggests that the heterogeneous evaluations create a lot of idle time for GE_{GEN}. The two algorithms perform similarly across the different parts. It is very difficult to distinguish between them, although perhaps it could be said that GE_{SS} performs slightly better at 500 evaluations on Part 1 and with 1,000 evaluations on Part 3, while GE_{GEN} performs a little bit better on Part 5 with 500 evaluations and parts 5 and 4 with 1,000 evaluations. On the Empirical Attainment Function plots, the median performance is similar for both algorithms and they often perform better in different regions. However, where there are differences this has mainly been shown in green, indicating a 20% - 40% difference in frequency. This difference, while important, occurs in less than half of the runs.

On parts 2, 4 and 5, GE_{GEN} finds better solutions in the region of the search space where the excess stock is low and the machining times are higher. These solutions are generally found by longer tool sequences that require more time to evaluate. On these same parts, GE_{SS} finds better solutions where the machining times are lower. Solutions on these parts take longer to evaluate, which is reflected in the runtimes of the algorithms. Figure 9.25 shows the frequency with which tool sequences of different lengths are evaluated by GE_{GEN} and GE_{SS}, when using 40 processors and 1,000 evaluations. Looking at this we can see if GE_{SS} is biased away from longer tool sequences. The first thing that we notice is that there is a great deal of variability between different runs for both algorithms. On Part 2, GE_{GEN} evaluates more 2, 3 and 5 tool sequences on average, while GE_{SS} evaluates more 4-tool sequences. On Part 4, the generational algorithm evaluates more 3 and 4 tool sequences, while GE_{SS} evaluates slightly more 5-tool sequences. On Part 5, GE_{GEN} evaluates more 3, 4 and 5-tool sequences. This shows that there are differences between the two algorithms but there is not a clear pattern that shows how solution-based heterogeneity may affect the distribution of tool sequence lengths evaluated by the algorithms. The length of the single-objective optimal solution also does not appear to affect the distribution. On Part 1, this solution contains two tools but the generational algorithm evaluates more sequences of this length on average. On Part 3, this solution contains 3 tools, and the asynchronous algorithm evaluates more solutions of this length, while GE_{GEN} evaluates more 2-tool sequences on average. It may be that caching reduces the solution-based heterogeneity.

The differences between the algorithms may be explained by their different search strategies. The generational algorithm, which uses enlarged child populations, could have a more exploratory search strategy. The asynchronous algorithm may exhibit more exploitative behaviour, due to the steady-state selection scheme (Durillo et al., 2008). The results in this chapter support previous work comparing synchronous and asynchronous versions of NSGA-II. As in (Durillo et al., 2008) and (Yagoubi et al., 2011), in this work the asynchronous algorithm was found to have a much faster running time. On many of the parts, the synchronous algorithm appears to achieve a slightly higher hypervolume score, which was also found in (Durillo et al., 2008). This differs from (Yagoubi et al., 2011), where the asynchronous algorithm was found to achieve a better hypervolume score. In (Yagoubi and Schoenauer, 2012) situations are shown where solution-based time heterogeneity can cause asynchronous algorithms to perform worse in certain regions of the Pareto front. On the larger parts (2, 4 and 5) the GE_{GEN} does seem to cover certain longer to evaluate regions better than the GE_{SS}. It would be interesting to implement *Duration-Based Selection*, introduced in (Yagoubi and Schoenauer, 2012), to see if this results in a change in regions covered by the steady-state search.

Given the results and the similarity between the two algorithms, it is likely that GE_{SS} would be more attractive in a real world setting. On Part 4, we see that the 40-processor version found the best result almost every time, in a fifth of the required machining time, when run for 1,000 evaluations.

In terms of reaching the assumed single-objective optimal solutions, both algorithms were only able to locate them every time on parts 1, 2 and 3, and only after 1,000 evaluations on the latter 2 parts. GE_{GEN} also located the single objective optimum on every trial on Part 5. A possible extension to the work in this chapter could be to implement an island model, such as in (Leon et al., 2008). For example, two islands could implement GE_{SS} with a 20-processor master/slave model on each, with migration of solutions between the islands. This could have the effect of increasing diversity and convergence on the optimal solution. Alternatively, different islands could use separate algorithms, perhaps with GE_{GEN} and GE_{SS} on different islands. This might improve the approximation of the Pareto front, as in many of the experiments we saw that the algorithms performed better in different regions.

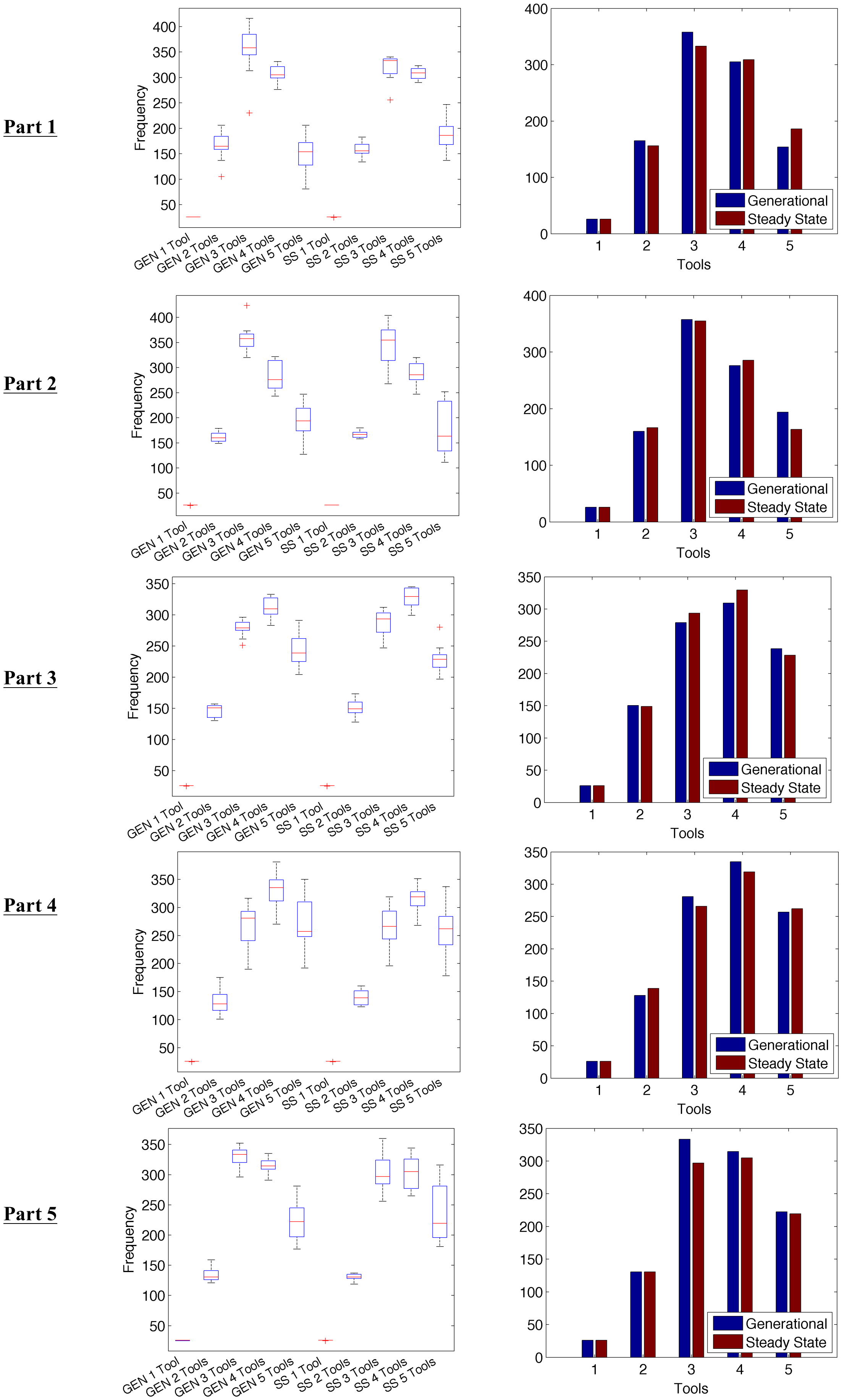
9.6 Summary

In this chapter we have investigated two ways that the master/slave parallel approach can be applied to the Tool Selection Problem using NSGA-II with Guided Elitism. Evaluating with real-time generation of tool paths and stock models across a number of trials, both synchronous generational and asynchronous steady-state search was found to be successful on all 5 of the tested parts. They were frequently able to obtain very good solutions and also returned a wide range of solutions offering a good trade-off between total machining time and excess stock that could be of interest to a decision maker. This suggests that the evolutionary approach works well on this problem, even when it is scaled up to a search space that is larger than the one used in previous chapters.

While both algorithms performed similarly, the generational algorithm, which employs an enlarged child population, often achieved a slightly better hypervolume score. While no regions were unattainable by the steady-state algorithm, this behaviour is interesting. It could be due to solution-based heterogeneity, although the algorithms found tool lengths with a similar distribution on average. Another reason for this could be additional exploratory capabilities in the generational algorithm, gained by sampling a large number of children at each generation.

The asynchronous algorithm was clearly superior in terms of runtime. It consistently took 30% – 40% less time than the synchronous generational algorithm. Combined with a very good search performance, it is likely that this approach would be more useful and relevant for an industrial setting. The system used in this chapter could easily be implemented on existing cloud or office grid computing services and so would be both accessible and relatively inexpensive for a workshop to apply.

Figure 9.25. On the left showing boxplots of the distribution of sequences of a certain lengths found over the course of the runs for GE_{GEN} (GEN) and GE_{SS} (SS) when run with 40 processors for 1,000 evaluations. On the right, bar charts are shown of the median scores.



Chapter 10

Conclusions and Further Work

10.1 Introduction

This thesis has primarily been concerned with the application of Evolutionary Computation to the optimisation of machining processes. The motivation behind this investigation has been to explore ways in which Evolutionary Algorithms can be incorporated into existing Computer Aided Manufacturing (CAM) software to aid process planners with decision making. Evolutionary Algorithms are flexible and extensible, meaning that they can be applied to many different types of optimisation problem. In this work they have been applied to both single and multi-objective problems, with variable length input vectors containing both discrete and continuous features.

In the experiments presented in previous chapters, we have seen that the evolutionary-powered systems can locate optimal tool sequences that minimise total machining time, as well as simultaneously optimising machining parameters to find trade-off solutions between time and either tooling costs or remaining material. Throughout this thesis novel contributions have been have been offered to both Evolutionary Computation and Machining. In Evolutionary Computation, new methods have been developed for preferential search and constraint handling in multi-objective Evolutionary Algorithms. In Machining, new metaheuristic operators and the first multi-objective Tool Sequence Optimisation systems have been introduced. Peer-reviewed published work extracted from this thesis has been referenced by researchers in both Evolutionary Computation (Segura et al., 2013) and Machining (Meng et al., 2014). The work in this thesis also offers many practical benefits to those involved with the Machining industry, such as automating tool sequence selection and providing a set of solutions that can deliver significant savings for both machining times and tooling costs.

This chapter is organised in the following way. Firstly, a chapter-by-chapter summary is given describing all of the work presented in this thesis. Secondly, the major contributions of this work are presented before discussion of a number of directions that could be taken for future work.

10.2 Summary

Reviewing the literature in Chapter 2, we discovered three main under-represented areas in optimisation in rough machining. The first is that most Tool Sequence Optimisation (TSO) work has concentrated on flat end mill tools, and none has supported the three main cutting tool geometries. The second is that there have been no multi-objective TSO systems, although there have been in Machining Parameter Optimisation (MPO). The third is that there has been very little work combining TSO and MPO, and none that has used a multi-objective approach. This thesis has mainly contributed to these three areas, as well as introducing a system for applying parallel multi-objective Evolutionary

Algorithms to TSO problems. We will now give a brief summary of the work that has been presented in each chapter.

As well as reviewing related research, in Chapter 2 the main principles behind machining and Evolutionary Computation are described, providing background to the inexperienced reader. In Chapter 3, the methods used throughout this thesis for evaluating tool sequences are introduced, and a system is presented that can both significantly reduce the amount of processing needed for evaluating tool sequences by caching tool sequence fragments, and also distribute processing across many computers.

This system is employed in Chapter 4 to exhaustively search through every solution on a constrained Tool Selection Problem using a part with a sculptured surface and a realistic tool library. Analysing the solutions found, two heuristics were discovered in the literature that could potentially lead to inefficient tool sequences being returned when using multiple tool types. The first of these is fixing the last tool in the sequence to the largest able to remove all material, which is found in (Lim et al., 2001; D'Souza et al., 2004; Ahmad et al., 2010). The heuristic did not hold in this case because it does not take cutting speeds into account. Depending toolpath lengths, a faster smaller tool can finish before a larger tool. A second heuristic, found in (D'Souza et al., 2004), suggests that the geometrical profile left by a larger tool is always the subset of that left by a smaller tool. Results from Chapter 4 found that this is not the case when using combinations of different tool types. Later work presented in (Ahmad et al., 2010), also found that this assumption does not hold when using tool holders, leading the authors to abandon graphical search methods and explore Genetic Algorithms.

A set of rules were developed in Chapter 4 that permit tool sequences to be generated with a free ordering, supporting flat end mill, toroidal and ballnose tools, while preventing redundancy. These are a logical extension to those presented in (D'Souza et al., 2004), supporting our new, less constrained model. With this came a new representation scheme, introduced in Chapter 5, deviating from the rigid structure used in (Ahmad et al., 2010; Chen and Fu, 2011) and allowing for a more flexible ordering. Two mutation operators were also compared, and a novel gradual method was found to significantly improve algorithmic performance in terms of both solution quality and run time. This area has been under represented in the literature, exemplified by this quote from Krimpenis and Vosniakos,

> "The implemented main GA is an off-the-shelf algorithm and its parameters take typical values according to the program manual." (2009: 455)

A contribution of this thesis is providing problem-specific operators that other researchers can integrate into their models. In Chapter 5 these operators are incorporated into four metaheuristic algorithms, a Genetic Algorithm (GA), Stochastic Hill Climber (SHC), Random Restart Stochastic Hill Climber (RRSHC) and a hybrid of the GA and SHC (HGA). The algorithms are tested on four parts, and display different strengths and weaknesses in terms of the quality of solutions and number of evaluations required. Faced with a search landscape containing many local optima, SHC and the HGA are significantly outperformed by RRSHC and the GA. This landscape was distorted by an error in the simulation process that hybridised two parts but served as a useful testing search space. RRSHC and the GA performed similarly but were unable to frequently locate the optimum solution using an acceptable number of evaluation functions.

One method that has been successful in overcoming premature convergence on suboptimal solutions is multi-objectivisation (Knowles et al., 2001). Here, multi-objective techniques are applied to single objective problems, which can help identify good building blocks and overcome problems caused by epistasis. Multi-Objective Optimisation (MOO) was used in this thesis both to improve performance on the difficult test search space presented in the Chapter 5, and to provide process planners with a selection of solutions, allowing them to make more informed decisions. The key concepts behind MOO are presented in Chapter 6, which could serve as a gentle introduction to the subject for those unfamiliar with the field. A novel modification to NSGA-II is introduced, *Guided Elitism*, that acts as a hybrid between single and multi-objective search. This is applied to the single-objective 0/1 Knapsack Problem, and is shown to outperform a single-objective GA and unconstrained NSGA-II.

In Chapter 7, TSO is multi-objectivised by treating the main constraint, excess stock, as a second objective. Multi-objective algorithms are able to efficiently and effectively explore the test search space that proved very difficult for single-objective techniques. The Guided Elitism and R-NSGA-II algorithms, which steer search towards the region of the search space with the desired surface tolerance, performed particularly well. In addition to frequently finding the solution optimal under the single-objective formulation, the algorithms return a diverse range of Pareto optimal solutions that provide a decision maker with several different options that they can choose from based on their current needs. While these approaches have been applied to TSO, they could prove useful to other researchers, as they show that multi-objectivisation with preferential search can provide efficient and effective search in the face of several local optima and sharp jumps in fitness.

For the most part, researchers have looked at MPO and TSO as separate problems. When tuning machining parameters, the tool (single-tool cases are almost exclusively looked at) is very often predetermined, while in TSO, machining parameters are almost always selected using manufacturers' recommendations. In addition, no TSO work reported in the literature has used a multi-objective approach.

The multi-objectivisation work in Chapter 7 provides the first exploration of multi-objective TSO, showing that desired solutions can be located in difficult search landscapes while providing the decision maker with a selection of trade-off solutions. This is extended in Chapter 8, where we introduce, for the first time, a truly multi-objective system. Here, two objectives are simultaneously optimised, total machining time and total tooling costs. Tool wear, an important factor in real world machining is considered in the tool sequence model. The most important machining parameter affecting tool wear, the cutting speed of a tool, is also included as a decision variable. This represents one of the few times that MPO has been combined with TSO and the first time this has been done in a multi-objective system.

As with the work in Chapter 5, in Chapter 8 excess material is treated as a hard constraint but included as an objective in the multi-objective search procedures. Different methods are used for constraint handling. These included the preferential search strategies used in Chapter 7, Guided Elitism and R-NSGA-II, a punishment factor, and a novel extension of (Deb, 1999), referred to as *Precedential Objective Ordering*. The multi-objective algorithms are shown to return a diverse selection of solutions, with Precedential Objective Ordering being particularly successful. The results of this chapter are an important contribution demonstrating two benefits of this system. Firstly, a multi-objective approach offers the decision maker many trade-off solutions between machining time and tooling costs, providing much more information than a priori weighted aggregate functions used in, for example, (Wang and Jawahir, 2005; Ahmad et al., 2010). Secondly, different tool sequences can allow different regions of the Pareto front to be reached. Within each tool sequence, optimising parameter values can locate a range of solutions. This suggests that it is important to consider TSO and MPO together in a multi-objective framework.

In the final results chapter, Chapter 9, a parallel master/slave model is introduced to deal with the problem of computationally expensive evaluations. Using a larger search space than in previous chapters, both synchronous generational (SYN) and asynchronous steady-state approaches (ASY) are implemented, running real simulations as opposed to working on cached sequences. This is designed to be easily implementable on existing grid services such as a cluster of heterogeneous computers or cloud web services. This makes it accessible to even the smallest of workshops. Potential problems for asynchronous steady-state parallel algorithms in the presence of solution-based evaluation time heterogeneity have been highlighted in (Yagoubi et al., 2011; Yagoubi and Schoenauer, 2012). With this in mind, the two master/slave approaches are carefully compared across five parts. The results show that the two algorithms perform similarly in terms of solution quality, although SYN slightly edges ASY in terms of finding a diverse range of trade-off solutions. However, ASY is significantly faster, which combined with a good search performance would make it likely to be the preferred algorithm in an industrial setting.

10.3 Contributions

This thesis has contributed to both Evolutionary Computation and Machining. In the area of Evolutionary Computation, it has been shown through numerous examples that single and multi-objective Evolutionary Algorithms are able to find optimal or near optimal solutions to Machining problems, using only a small number of evaluations. This is accomplished largely through the use of small population sizes, a less routinely applied method of deployment. It was shown in Chapter 7 that the normal operation of the NSGA-II algorithm can lead to poor performance with small population sizes, and a novel modification was introduced to avoid this. Multi-objectivisation is a small subfield of Multi-objective Optimisation, and the use of preferential search in it has been underrepresented. It has been shown in this thesis that preferential search can improve the efficiency and effectiveness of multi-objectivised single-objective optimisation using a tight budget of evaluations, and a novel method for preferences was introduced, which has been referenced by (Segura et al., 2013). To the best of the author’s knowledge, the work presented in Chapter 9 is the first study in the literature that has evaluated differences between multi-objective synchronous and asynchronous parallel algorithms that incorporate preferences. A final contribution to Evolutionary Computation is the introduction of a new method for handling constraints in multi-objective problems, which was found to perform best in the experiments in Chapter 8 and could prove useful to other researchers.

In the area of Machining there have been many contributions that are of both academic and practical interest. Academically, heuristics used in previous TSO work have been shown to prevent optimal solutions from being found. Additionally, new operators have been introduced for applying metaheuristic algorithms to TSO problems, which are shown to improve search and support a wider range of tool types. In particular, we have demonstrated that using a small number of evaluations we can find sequences with low machining times, which has been referenced by (Meng et al., 2014). In Chapters 7, 8 and 9, multi-objective techniques are applied to TSO and MPO for the first time, to the best of the author's knowledge. The work in Chapter 8 adds to the literature by being only the fourth study combining TSO and MPO, and the first to use multi-objective techniques. The work in this thesis has added to the academic study of optimisation in Machining and presents both interesting results and also several further directions that the research can take in the future, which are discussed in greater detail in the next section.

For those involved with the Machining industry, this work offers several practical benefits. It is a step towards the full automation of machining, reducing the number of hours that need to be spent on planning by engineers. In the single-objective work, which focused solely on minimising time, finding optimal solutions can produce large machining time savings compared to using the "greedy" method described in (Lim et al., 2001; Bouaziz and Zghal, 2008). For example, using the tool library in Chapter 4, 225 minutes (19%) can be saved on Part 2 and 378 minutes (21%) can be saved on Part 4 when using the optimal solution. These are significant savings that can dramatically reduce the cost of machining. Given a base rate of £50p/h to machine in a workshop, this is a saving of £187.5 and £315 respectively on the two parts. A company producing 200 parts similar to these in a year could save more than £60,000 through using efficient machining processes.

Additionally, through the simultaneous optimisation of tool sequences and cutting speeds, significant savings can be made both in terms of machining time and tooling costs. Using Part 4 as an example, the "greedy method" combined with handbook parameter values produces a sequence that requires 1,229 minutes to complete with tooling costs of £2,759, when using the library described in Chapter 8. Keeping the machining time the same, the multi-objective system described in Chapter 8 can produce the same part for tooling costs of around £600, less than a quarter of the cost. If 50 of these parts were produced in a year, over £100,000 could be saved in tooling costs. For a similar cost, the system finds a sequence and corresponding parameters that requires around 750 minutes of machining time, a saving of almost 9 hours. The results presented in Chapter 8 show that many solutions can be found offering trade-offs between machining times and tooling costs. This could prove useful for process planners who, for example, would like to reduce the time taken to produce two parts, in order to manufacture them both in a single day. It also provides a selection of tool sequences that can achieve similar objective values, which gives planners options. For example, they may decide to use one sequence over another because it makes use of tools that the workshop has a large stock of, or decide against a sequence because the same tools are needed for the next job.

In summary, work in this thesis has contributed academically to Evolutionary Computation, with new methods for applying preferences and constraints to multi-objective systems with small population sizes, and to Machining by offering new metaheuristic operators for TSO and the first multi-objective TSO, and TSO with MPO systems. Finally, this work offers many practical benefits to both producers and consumers in the Machining industry, such as the ability to automate the generation of tool sequences and find optimal sequences that can provide large reductions in machining time, as well as a selection of solutions offering trade offs between machining times and tooling costs.

10.4 Further Directions

There are many interesting directions that the work presented here could take in the future. These include further development of the models presented by exploring new algorithms and search techniques to improve effectiveness and efficiency; integrating extra parameters and objectives into the existing system; and methods to evaluate a larger number of solutions.

10.4.1 Developing existing systems

There were two main systems presented in this thesis:

(1) A single-objective tool sequence optimisation system

(2) A multi-objective tool sequence and cutting speed optimisation system

Below we will discuss ways that these systems could be extended, as well as additional objectives that could be introduced.

10.4.1.1 Single Objective System

In the first system, four single-objective metaheuristic algorithms and a number of variants of NSGA-II were used to power the search. The NSGA-II versions operated on a “multi-objectivised” formulation of the problem, where excess stock was treated as a soft constraint and included as an objective. Ignoring the added benefits of multi-objective search in terms of returning a selection of solutions, the goal of the system is to find the solution with the shortest machining time satisfying the surface tolerance constraint.

Future development of these systems should aim to improve on the ability of the single-objective algorithms to locate optimal solutions in the presence of rugged search landscapes and the multi-objective algorithms to reduce the number of evaluations needed, especially on parts with smoother search landscapes. Given the propensity of Hill Climbers to end up in local optima, methods based on Simulated Annealing (SA) may lead to better solutions (Bertsimas and Tsitsiklis, 1993). SA works in a similar way to SHC but has a probability of accepting a worse solution that decreases as search progresses. This allows it to move out of local optima. It would be interesting to apply SA techniques to TSO, incorporating the representation and exploration methods introduced in Chapter 5.

Simulated Annealing could also prove useful for hybridisation. The hybrid Genetic Algorithm (HGA) used in Chapter 5, which incorporated SHC, greatly increased the GA’s convergence speed but contributed to poor performance in a rugged search landscape. A Genetic Algorithm-Simulated Annealing hybrid could potentially overcome performance problems, and has been shown to perform well in MPO in (Wang et al., 2005b). Additionally, it would be interesting to move from Lamarckian evolution, where local search changes modify the genotype, to Baldwinian evolution, where changes are not passed on. This could help prevent premature convergence.

Local search could also be included in the multi-objective algorithms, through a serial or concurrent approach (Sindhya et al., 2011). In the first, local search is performed on non-dominated solutions returned by the multi-objective algorithms, after the initial global search has finished. In the second, local search is applied to solutions returned during the global search. Again SA could be used to seek out local improvement, which has been modified to use Pareto dominance in (Smith et al., 2004), as could the Hill Climbing-like technique used in M-PAES (Knowles and Corne, 2000).

It was seen in Chapter 7 that both Guided Elitism and R-NSGA-II performed well with small population sizes. This contributed to them producing good results with low evaluation limits. A problem with using small population sizes in Evolutionary Algorithms is that there are a reduced number of building blocks kept in population memory (Ahn and Ramakrishna, 2003). This can be exacerbated by a lack of diversity. The use of strong elitism in NSGA-II maintains selection pressure, which was shown to be important for micro-GAs in (Ahn and Ramakrishna, 2003), while Crowding Distance Assignment (CDA) helps maintain diverse solutions. The effectiveness of the algorithms could potentially be improved by using an external memory (archive) to hold solutions. In (Coello Coello and Toscano, 2001; Toscano and Coello Coello, 2003) an external archive is kept of replaceable and irreplaceable solutions. The irreplaceable solutions are generated in an initial population and help maintain diversity in the presence of small populations. In Pareto Archiving Evolutionary Strategy (PAES), a large external archive of solutions is kept which uses similar principles to CDA to maintain a diverse population (Knowles and Corne, 1999). Incorporating preferences in these algorithms, or an archiving system inspired by them using reference points or Guided Elitism, could increase search success while maintaining low computational cost on tool sequence optimisation problems.

In Chapter 7 it was seen that R-NSGA-II performed very well with some reference points but was less successful with others. Its top configuration was the best performing of all the algorithms tested. It would be interesting to test the effect of using multiple reference points, or co-evolving single or multiple reference points in a similar way to PICEA-g (Wang et al., 2013). Creating a parallel model for the R-NSGA-II could also prove fruitful. One possibility is creating a heterogeneous island model, similar to (Leon et al., 2008) where different islands use different reference points. In this vein, there are further island models that could be explored, such as having different algorithms, such as NSGA-II, R-NSGA-II and Guided Elitism all running on separate islands with migration. This could enable a good spread of solutions to be returned, while also focusing on the desired region. One problem that would have to be addressed would be handling increased evaluation costs, which could perhaps be avoided through employing smaller population sizes than those used in Chapter 9. It is possible that running four algorithmically heterogeneous islands with smaller population sizes, using a master/slave system on each island, produces better results than running only Guided Elitism with a larger population on a single island.

10.4.1.2 Multi-objective System

Hybridisation and parallelisation could also benefit the multi-objective system presented in Chapter 8. Local search could be applied to the discrete tool sequences by taking a solution and searching through neighbours a set number of mutations away, terminating if a solution is found that dominates the original. This could be continued to a set depth (locating several solutions that dominate the previous best), and could use Hill Climbing or SA. At the same time, after every discrete mutation, it will be necessary to perform some local search on the continuous parameters. One option for this is (1+1)-Covariance-Matrix-Adaptation-Evolutionary Strategy (Igel et al., 2006), which has been shown to be a very high performing continuous optimiser (Hansen et al., 2010).

In this same vein, local search could be applied to only the continuous parameters, i.e. operating only on existing tool sequences. At each generation, for each tool sequence and cutting speed solution in the population, a continuous optimiser could be used to improve the continuous parameters and find solutions that dominate the original. Another extension could be to explicitly divide the system presented in Chapter 8 into a two-stage process, using multiple algorithms. The first stage is concerned with finding tool sequences located in different regions of the Pareto front. This could be performed by perhaps dividing the regions into islands and using the R-NSGA-II with different reference points in each island, as discussed above, or having Precedential Objective Ordering, unconstrained NSGA-II and versions of R-NSGA-II with different reference points all on separate islands, with migration. At the end of the first stage, defined by a function evaluation limit, the solutions from the different islands are combined and ranked using non-dominated sorting. Each non-dominated solution then has multi-objective optimisation performed on the cutting speeds using a continuous multi-objective population-based algorithm. For example, the Multi-Objective Covariance Matrix Adaptation Evolutionary Strategy (MO-CMA-ES) (Igel et al., 2007) or the multi-objective Dynamic Multi-Swarm Particle Swarm Optimizer (Liang et al., 2012) could be used, each of which has been shown to outperform NSGA-II on continuous optimisation problems (Voß et al., 2010; Liang et al., 2012). This could help provide a large and diverse range of solutions to decision makers. However, work would have to be done to ensure that function evaluations are kept to a level that would make this applicable to real world situations.

10.4.1.3 Additional Objectives

Both the multi-objectivised TSO system presented in Chapters 7 and 9 and the multi-objective TSO and MPO system presented in Chapter 8 could be extended to include additional objectives. The goal of this would be to provide additional information to decision makers. In terms of providing options for discrete tool sequences, one objective could be a measure of the amount of work done by larger tools. For example, for each tool in the sequence, the tool path could be divided by the tool diameter. The sum of this over all tools would be the objective function that we would aim to minimise. Another objective could be surface uniformity, the average amount of excess stock on the surface of the machined part, as used by (Krimpenis and Vosniakos, 2009). In addition to these, in the system presented in Chapter 8 we could also include the number of tools used as a new objective (as a tool needs to be replaced if it is too worn), to minimise the need to have access to large numbers of tools.

As discussed in Chapter 6, Pareto-based multi-objective algorithms such as NSGA-II perform badly with more than three objectives, as there is a large increase in the number of non-dominated solutions. For this reason, when including more objectives better success may be had with PICEA or MOEA/D (Zhang and Li, 2007).

10.4.2 Machining Optimisation: A unified approach

Very little research has combined TSO and MPO. The results in Chapter 8 show its promise, as different discrete tool sequences can locate alternative parts of the Pareto front. Future work should look to further unify the two research areas by including more machining parameters. For example, research found in (Sardiñas et al., 2006, Datta and Deb, 2009; Solimanpur and Ranjdoostfard, 2008) all use cutting speed, depth of cut (DOC) and feed. Incorporating the latter two parameters for each tool in a sequence would offer the ability to perform more efficient machining. (Krimpenis and Vosniakos, 2009) offer the most complete system in this respect, supporting a large number of machining parameters, but do not include multiple tools, only single tool selection from a library of 8 tools.

Unlike cutting speed and feed, changes in DOC modifies tool paths. This means that for each unique DOC, a new set of tool paths and stock models need to be generated in the CAM software, at added computational cost. This is exemplified in (Wang et al., 2005b), where cutting speed and feed are optimised using four fixed DOCs, in individual runs. However, DOC itself is not included in the decision vector. In Chapter 8’s model, many more cutting speed evaluations were allowed than tool sequence modifications because of their much shorter assessment times. Discretising DOC values so they could only be moved in steps of 0.2 or 0.5mm and limiting the amount that they can deviate from manufacturers’ recommendations could help to stem the combinatorial explosion somewhat but the search space would still increase greatly. For example, if we only consider a search space of 100 legal two-tool combinations, adding DOC (and disregarding any other parameters) with just 10 mutable discrete options would increase the number of possible combinations from 100 to 10,000. One of the advantages of evolutionary techniques is their ability to find good solutions in large complex search spaces, when sampling only a small part of the search space. Therefore, it is likely that the approach presented in this thesis could be extended to finding useful solutions to this larger, more complex problem, the caveat being that without doubt, it would be necessary to increase the number of expensive simulations needed, while simultaneously finding methods to limit the number required.

10.4.3 Approximating Fitness Evaluations

There have been many approaches to approximating expensive evaluation functions in single-objective (Jin, 2005) and multi-objective optimisation (Santana-Quintero, 2010). This could be useful for including DOC as an additional decision variable and also for speeding up the single and multi-objective systems explored in this thesis. One good candidate could be *Fitness Inheritance* (Smith et al., 1995), which has been shown to provide a large reduction in evaluations in both single and multi-objective optimisation (Smith et al., 1995; Chen et al., 2002; Bui et al., 2005; Barbour et al, 2010). Here, at each generation, a proportion of individuals are assigned fitness through the evaluation function, while the rest receive an averaged score based on the fitness of their parents. In TSO, it is possible that this could be improved by including problem-specific knowledge and also using the nearest neighbours of the solution in fitness estimation (Branke and Schmidt, 2005).

A second option is the use of *Surrogate Approximation* (Santana-Quintero et al., 2010). Here, the evaluation function is approximated through the use of an online, learning model such as an Artificial Neural Network (ANN) or Support Vector Machine (SVM). In (Voutchkov and Keane, 2006) NSGA-II is combined with a surrogate. Initially 20 solutions are evaluated by simulation and a model is trained. The algorithm then evolves for 50 generations with a population of 50, using the surrogate for evaluation. 20 equally spaced solutions are chosen and are evaluated using the real function. These are added to the pool of simulations and the model is retrained. This process repeats 20 times.

In (Nain and Deb, 2005), an ANN is integrated with NSGA-II. For a set number of generations, all solutions are evaluated by the exact fitness function, and following this the ANN is trained on the preceding data. In the subsequent predefined number of generations, all solutions are evaluated with the ANN surrogate. This process is repeated until the end of search. An important extension to the work presented in this thesis would be to experiment with surrogate approximation but the surrogate model would have to be carefully constructed, given the mix of discrete and continuous parameters in the system presented in Chapter 8.

Fitness Inheritance and Surrogate Approximation could both help reduce the number of expensive evaluations that need to be employed. It is possible that a combination of the two, which carefully selects which solutions to approximate, could produce an efficient search scheme. One potential issue that would have to be tested is the added noise that comes from using fitness imprecision. It was seen in Chapters 5 and 7 that the single-objective algorithms performed badly when presented with a search space with several local optima and although this was much improved using multi-objectivisation, it will need to be carefully evaluated.

10.5 Closing Remarks

In this thesis we have contributed to under-represented areas in rough machining optimisation, using single and multi-objective approaches. We have created systems that support the use of real world tools and that can find near-optimal solutions in reasonable amounts of time, using computing resources available to even small workshops. We have also produced the first multi-objective Tool Sequence Optimisation system and the first system to combine Tool Sequence Optimisation and Machining Parameter Optimisation using a multi-objective approach. Additionally, we have developed techniques that provide preferences and constraint handling in multi-objective search, while returning a diverse range of solutions that we hope will be useful in other applications.

We believe that the multi-objective approach is an important step forward, as it can offer much more information to decision makers, which enables them to make a more informed choice based on their personal circumstances. Thus, we hope that the contributions in this thesis will be integrated into industrial Computer Aided Manufacturing software and assist engineers in finding efficient rough machining strategies.

Bibliography

Ahmad, Z., Rahmani, K., and D'Souza, R. M., *Applications of genetic algorithms in process planning: tool sequence selection for 2.5-axis pocket machining*. Journal of Intelligent Manufacturing, *21*(4), pp. 461-470, 2010.

Ahn, C. W., and Ramakrishna, R. S., *Elitism-based compact genetic algorithms*, IEEE Transactions on Evolutionary Computation, 7(4), pp. 367–385, 2003.

Ahn, C. W., and Ramakrishna, R. S., *Elitism-based compact genetic algorithms*, IEEE Transactions on Evolutionary Computation*,* 7(4), pp. 367-385, 2003.

Andre, J., Siarry, P., and Dognon, T., *An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization*, Advances in engineering software, *32*(1), pp. 49-60, 2007.

Astakhov, V. P. and Davim, P. J., *Tools (Geometry and and Material) and Tool Wear*, in Machining Fundementals and Recent Advances (ed. J.P. Davim), Springer London, 2008.

Auger, A., Bader, J., Brockhoff, D. and Zitzler, E, Articulating user preferences in many-objective problems by sampling the weighted hypervolume. In Proc. 11th Annual conference on Genetic and evolutionary computation (GECCO'09), pp. 555-562, 2009.

Barbour, R., Corne, D., and McCall, J., *Accelerated optimisation of chemotherapy dose schedules using fitness inheritance*, In 2010 IEEE Congress Evolutionary Computation (CEC2010), pp. 1-8, 2010.

Baskar, N., Asokan, P., Saravanan, R., and Prabhaharan, G., *Selection of optimal machining parameters for multi-tool milling operations using a memetic algorithm*, Journal of Materials processing technology, 174(1), pp. 239-249, 2006.

Berrichi, A., Amodeo, L., Yalaoui, F., Châtelet, E., & Mezghiche, M., *Bi-objective optimization algorithms for joint production and maintenance scheduling: application to the parallel machine problem*, Journal of Intelligent Manufacturing, *20*(4), 389-400, 2009.

Bertsimas, D., and Tsitsiklis, J., *Simulated annealing*, Statistical Science, 8(1), pp. 10-15, 1993.

Beume, N., Naujoks, B., and Emmerich, M., *SMS-EMOA: Multiobjective selection based on dominated hypervolume*, European Journal of Operational Research, *181*(3), 1653-1669, 2007.

Beyer, H. G., and Schwefel, H. P., Evolution strategies–A comprehensive introduction. *Natural computing*, *1*(1), pp. 3-52, 2002.

Black, J.T. and Kohser, R.A., *DeGarmo's Materials and processes in manufacturing* (11th edition), Wiley, NJ, USA, 2012.

Blum, C. and Roli, A. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*. ACM Comput. Surv, 35(3), pp. 268-308, 2003.

Bouaziz, Z., and Zghal, A, Optimization and selection of cutters for 3D pocket machining. *International Journal of Computer Integrated Manufacturing*, *21*(1), pp. 73-88, 2008.

Branke, J. and Deb, K, *Integrating user preferences into evolutionary multi-objective optimization*. Knowledge Incorporation in Evolutionary Computation (ed. Y. Jin), pp. 461–477, Springer: Hiedelberg, Germany, 2004.

Branke, J., and Schmidt, C., *Faster convergence by means of fitness estimation*, Soft Computing, 9(1), pp. 13-20, 2005.

Branke, J., Kaußler, T. and Schmeck, H, *Guidance in evolutionary multi-objective optimization*, Advances in Engineering Software, 32, pp. 499–507, 2001.

Bui, L. T., Abbass, H. A., and Essam, D., *Fitness inheritance for noisy evolutionary multi-objective optimization*. In Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO'05), pp. 779-785, 2005.

Cantú-Paz, E., *A survey of parallel genetic algorithms*. Calculateurs paralleles, reseaux et systems repartis, *10*(2), pp. 141-171, 1998.

Cantu-Paz, E., *Efficient and accurate parallel genetic algorithms*, Springer New York, 2000.

Carpenter, I.D. and Maropoulos, P.G., *Automatic tool selection for milling operations Part 1: cutting data generation*, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 214(4), pp. 271-282, 2000(a)

Carpenter, I.D. and Maropoulos, P.G., *Automatic tool selection for milling operations Part 2: tool sorting and variety reduction*, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 214(4), pp. 283-292, 2000(b).

Ceroni, A., Pelikan, M., and Goldberg, D. E., *Convergence-time models for the simple genetic algorithm with finite population*, IlliGAL Report No. 2001028, Urbana, IL: Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2001.

Chen, J. H., Goldberg, D. E., Ho, S. Y., and Sastry, K., *Fitness Inheritance In Multi-objective Optimization*. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02), pp. 319–326, 2002.

Chen, X., Ong, Y. S., Lim, M. H., and Tan, K. C., *A multi-facet survey on memetic computation*, IEEE Transactions on Evolutionary Computation, *15*(5), 591-607, 2011.

Chen, Z. C., and Fu, Q., *An optimal approach to multiple tool selection and their numerical control path generation for aggressive rough machining of pockets with free-form boundaries*, *Computer-Aided Design*, *43*(6), pp. 651-663, 2011.

Churchill, A. W., Husbands, P. and Philippides, A., *Metaheuristic approaches to tool selection optimisation*, in Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO '12), pp. 1079-1086, 2012.

Churchill, A. W., Husbands, P., and Philippides, A., *Multi-objectivization of the Tool Selection Problem on a Budget of Evaluations*, in Proceedings of the seventh conference on Evolutionary Multi-Criterion Optimization (EMO 2013), pp. 600-614, 2013a.

Churchill, A. W., Husbands, P., and Philippides, A., *Multi-objective tool sequence and parameter optimization for rough milling applications*, in IEEE Congress on Evolutionary Computation (CEC) 2013, pp. 1475-1482, 2013b.

Churchill, A. W., Husbands, P., and Philippides, A., Tool sequence optimisation using preferential multi-objective search. in Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion, pp. 181-182, 2013c.

Churchill, A. W., Husbands, P., & Philippides, A. *Tool sequence optimization using synchronous and asynchronous parallel multi-objective evolutionary algorithms with heterogeneous evaluations*. in IEEE Congress on Evolutionary Computation (CEC) 2013, pp. 2924-2931, 2013d.

Coello Coello, C.A., and Toscano G.P., *A micro-genetic algorithm for multiobjective optimization*, in Proc. 1st Int. Conf. Evolutionary Multi-Criterion Optimization, pp. 126-140, Springer Berlin, Heidelberg, 2001.

Coello Coello, C. A., and Lamont, G. B., *Applications of Multi-Objective Evolutionary Algorithms*, World Scientific Pub Co. Pte. Ltd. Hackensack, NJ, 2004.

Coello Coello, C.A., Van Veldhuizen, D.A. and Lamont, G.B. *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd edition, Springer New York, NY, 2007.

Crowson, R., *Parts Fabrication: Principles and Process (Handbook of Manufacturing Engineering, Second Edition)*, CRC Press, UK, 2006.

Datta, R., and Deb, K., *A classical-cum-evolutionary multi-objective optimization for optimal machining parameters*, World Congress on Nature & Biologically Inspired Computing, 2009 (NaBIC 2009), pp. 607-612, 2009.

Davis, L., *Job Shop Scheduling with Genetic Algorithms*, Proceedings of the 1st International Conference on Genetic Algorithms, pp.136-140, 1985.

Deb, K., *Multi-objective optimization using evolutionary algorithms*, Chichester: John Wiley & Sons, 2001.

Deb, K., and Agrawal, S., A niched-penalty approach for constraint handling in genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*, pp. 235-243, 1999.

Deb, K. and Goldberg, D.E, *An Investigation of Niche and Species Formation in Genetic Function Optimization*, Proceedings of the Third International Conference on Genetic Algorithms, pp. 42-50, 1989.

Deb, K. and Sundar, J., *Reference point based multi-objective optimization using evolutionary algorithms*, In Proc. 8th Annual Conference on Genetic and Evolutionary Computation Conference (GECCO '06), pp. 635-642, 2006.

Deb, K., and Goel, T., *Controlled elitist non-dominated sorting genetic algorithms for better convergence*. In *Evolutionary Multi-Criterion Optimization*, pp. 67-81, Springer Heidelberg, Berlin, 2001.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), pp. 182-197, 2002.

Deb, K., and Saha, A, *Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach*, in Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO'10), pp. 447-454, 2010.

Deb, K., Zope, P. and Jain, A., *Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms*, Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), pp. 534–549, 2003.

di Pierro, F., Khu, S. T., and Savic, D. A., *An investigation on preference order ranking scheme for multiobjective evolutionary optimization*, IEEE Transactions on Evolutionary Computation, 11(1), pp. 17-45, 2007.

D'Souza, R. M., Sequin, C., and Wright, P. K., *Automated tool sequence selection for 3-axis machining of free-form pockets*, Computer-Aided Design, *36*(7), 595-605, 2004.

Durillo, J. J., Nebro, A. J., Luna, F., and Alba, E., *A study of master-slave approaches to parallelize NSGA-II*, In IEEE International Symposium on Parallel and Distributed Processing, 2008 (IPDPS 2008), pp. 1-8, 2008.

Fernando, C., Szathmary, E. and Husbands, P. Selectionist and evolutionary approaches to brain function: a critical appraisal, Frontiers in Computational Neuroscience, 2012.

Fonseca, C. and Fleming, P., *Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 28(1), pp. 26–37, 1998.

Fonseca, C.M. and Fleming, P.J. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416-423, 1993.

Forrest, S., and Mitchell, M., *Relative Building-block fitness and the Building-block Hypothesis, in* Foundations of Genetic Algorithms 2*,* Morgan Kaufmann, San Mateo, CA, 1993.

Friedrich, T., Kroeger, T. and Neumann, F., *Weighted preferences in evolutionary multi-objective optimization*, In Proceedings of the 24th international conference on Advances in Artificial Intelligence (AI'11), pp. 291-300, 2011.

Garza-Fabre, M., Toscano-Pulido, G. and Rodriguez-Tello, E., *Locality-based multiobjectivization for the HP model of protein structure prediction*, In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO '12), 473-480, 2012.

Giagkiozis, I., Purshouse, R. C., and Fleming, P. J, *An overview of population-based algorithms for multi-objective optimisation*, International Journal of Systems Science, pp. 1-28, 2013.

Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

Groover, M.P., Fundementals of Modern Manufacturing: Materials, Processes and Systems, 4th ed., Wiley, Hoboken, NJ, 2010.

Grosan, C., Improving the performance of evolutionary algorithms for the multiobjective 0/1 knapsack problem using ε-dominance. IEEE Congress on Evolutionary Computation, 2004 (CEC2004), pp. 1958-1963, 2004.

Hadka, D., and Reed, P., *Borg: An auto-adaptive many-objective evolutionary computing framework*, IEEE Transactions on Evolutionary computation, *21*(2), pp. 231-259, 2013.

Hansen, N., Auger, A., Ros, R., Finck, S. and Posik, P., *Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009*. Workshop Proceedings of the Genetic and Evolutionary Computation Conference 2010, ACM, 2010.

Hoos, H.H., *On the run-time behaviour of stochastic local search algorithms for SAT*, Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference, pp. 661-666, 1999.

Horn, J., Nafpliotis, N. and Goldberg, D.E, *A Niched Pareto Genetic Algorithm for Multiobjective Optimization*, In Proceedings of the First IEEE Conference on Evolutionary Computation, 1, pp. 82-87, 1994.

Houck, C., Joines, J. and Kay, M. *Utilizing Lamarckian evolution and the Baldwin effect in hybrid genetic algorithms*, NCSU-IE Technical Report, 1996.

Husbands, P. and Mill, F., *Simulated Co-Evolution as The Mechanism for Emergent Planning and Scheduling*, Proc. 4th Int. Conf. on Genetic Algorithms, pp. 264-270, 1991.

Husbands, P., *Genetic algorithms for scheduling*, AISB Quarterly, *89*, pp. 38-45, 1994.

Igel, C., Hansen, N. and Roth, S., *Covariance matrix adaptation for multi-objective optimization*, Evolutionary Computation, Vol. 15(1), pp.1-28, 2007.

Igel, C., Suttorp, T., and Hansen, N., *A computational efficient covariance matrix update and a (1+ 1)-CMA for evolution strategies*. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO'08), pp. 453-460, 2006.

Industrial Tool Corporation, ITC Catalogue Issue 8, http://www.itc-ltd.co.uk/downloads.htm (accessed 1/9/2012)

Ishibuchi, H. and Nojima, Y, *Optimization of scalarizing functions through evolutionary multiobjective optimization*, Evolutionary Multi-Criterion Optimization - EMO 2007, pp. 51-65, 2007.

Ishibuchi, H. and Nojima, Y., *Optimization of scalarizing functions through evolutionary multiobjective optimization*, Lecture Notes in Computer Science 4403: Evolutionary Multi-Criterion Optimization - EMO 2007, pp. 51-65, 2007.

Ishibuchi, H., Doi, T., and Nojima, Y., *Incorporation of scalarizing fitness functions into evolutionary multiobjective optimization algorithms*. Parallel Problem Solving from Nature-PPSN IX, pp. 493-502, 2006.

Ishibuchi, H., Tsukamoto, N., Sakane, Y., and Nojima, Y. *Indicator-based evolutionary algorithm with hypervolume approximation by achievement scalarizing functions*. In Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO'10), pp. 527-534, 2010.

Jaimes, A., and Coello, C., *Applications of Parallel Platforms and Models in Evolutionary Multi-Objective Optimization*, In Biologically-Inspired Optimisation Methods: Parallel Algorithms, Systems and Applications, pp. 23-49, Springer Berlin, Heidelberg, 2009.

Jensen, M. T., *Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation*, Journal of Mathematical Modelling and Algorithms, *3*(4), 323-347, 2004.

Jin, Y., *A comprehensive survey of fitness approximation in evolutionary computation*, Soft computing, 9(1), pp. 3-12, 2005.

Juels, A. and Wattenberg, M., Stochastic *Hillclimbing as a Baseline Method for Evaluating Genetic Algorithms*. University of California - Berkeley, Technical Report, CSD-94-834, 1994.

Juneja, B. L., and Seth, N., *Fundamentals of metal cutting and machine tools*. New Age International Daryaganj, New Delhi, 2003.

Kalpakjian, S. and S.R. Schmid, *Manufacturing Processes for Engineering Materials*, 2008.

Knowles, J. D., and Corne, D. W., M-*PAES: A memetic algorithm for multiobjective optimization*, In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000),* pp. 325-332, 2000.

Knowles, J., and Corne, D., *The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation*. In Proceedings of the 1999 Congress on Evolutionary Computation (CEC '99), 1999.

Knowles, J.D. and Corne, D.W. *Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy*, *Evolutionary Computation*, 8(2), pp. 149–172, 2000.

Knowles, J.D., Watson, R.A., and Corne, D.W., *Reducing local optima in single-objective problems by multi-objectivization*, In Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, pp. 269-283, 2001.

Knowles, J.D., Watson, R.A., Corne, D.W, *Reducing local optima in single-objective problems by multi-objectivization*, In Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, pp. 269-283, 2001.

Krimpenis, A., and Vosniakos, G. C., *Rough milling optimisation for parts with sculptured surfaces using genetic algorithms in a Stackelberg game*, Journal of Intelligent Manufacturing, 20(4), 447-461, 2009.

Lehman, J., Stanley, K.O. and Miikkulainen, R., *Effective diversity maintenance in deceptive domains*, In Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference (GECCO '13), pp. 215-222, 2013.

Leon, C., Miranda, G., and Segura, C., *Parallel hyperheuristic: a self-adaptive island-based model for multi-objective optimization*. In Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08), pp. 757-758, 2008.

Liang, J. J., Qu, B. Y., Suganthan, P. N., and Niu, B., *Dynamic multi-swarm particle swarm optimization for multi-objective optimization problems*. In 2012 IEEE Congress on Evolutionary Computation (CEC2012), pp. 1-8, 2012.

Lim, T., Corney, J., Ritchie, J. M., and Clark, D. E. R., *Optimizing tool selection*, International Journal of Production Research, *39*(6), pp. 1239-1256, 2001.

Lin, A. C. and Gian, R, *A Multiple-Tool Approach to Rough Machining of Sculptured Surfaces*, The International Journal of Advanced Manufacturing Technology, 15(6), pp.387-398, 1999.

Llorà, X., Reddy, R., Matesic, B., and Bhargava, R., *Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging*, In Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 2098-2105, 2007.

Lochtefeld, D.F. and Ciarallo, F.W., *Multiobjectivization via Helper-Objectives With the Tunable Objectives Problem*, IEEE Transactions on Evolutionary Computation, 16(3), pp. 373-390, 2012.

López-Ibáñez, M., Stützle, T., and Paquete, L., *Graphical tools for the analysis of bi-objective optimization algorithms:[workshop on theoretical aspects of evolutionary multiobjective optimization]*, In Proceedings of the 12th annual conference companion on Genetic and evolutionary computation (GECCO'10), pp. 1959-1962, 2010.

Luke, S. *Essentials of Metaheuristics*, Lulu, 2013, available at
http://cs.gmu.edu/~sean/book/metaheuristics/

Menczer, F., Degeratu, M., and Street, W. N., Efficient and scalable pareto optimization by evolutionary local selection algorithms. *Evolutionary computation*, *8*(2), pp. 223-247, 2000.

Meng, F. J., Chen, Z. T., Xu, R. F., & Li, X., *Optimal barrel cutter selection for the CNC machining of blisk*, Computer-Aided Design, 2014.

Michalewicz, Z. *Genetic algorithms + data structures = evolution programs*. Springer-Verlang, Berlin Heidelberg. 1998.

Michalewicz, Z., and Arabas, J., *Genetic algorithms for the 0/1 knapsack problem*, in 8th International Symposium on Methodologies for Intelligent Systems, pp.134-143, 1994.

Mitchell, M., *An introduction to genetic algorithms*, Cambridge, Massachusetts London, England, Fifth printing, 1999.

MSC Industrial Supply Co., http://www.mscdirect.co.uk/Milling/1375.html (accessed on 9/1/2013)

Nain P.K.S. and Deb, K., *A multi-objective optimization procedure with successive approximate models*, KanGAL report no. 2005002, Indian Institute of Technology, Kanpur, India, 2005.

Naudts, B., and Verschoren, A., *Epistasis and deceptivity*, Bulletin of the Belgian Mathematical Society, 6, pp. 147-154, 1999.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E., *MOCell: A cellular genetic algorithm for multiobjective optimization*, International Journal of Intelligent Systems, 24(7), pp. 726-746, 2009.

Nguyen, H. D., Yoshihara, I., Yamamori, K., and Yasunaga, M., *Implementation of an effective hybrid GA for large-scale traveling salesman problems*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 37(1), pp. 92-99, 2007.

Oberg, E., Jones, F. D., Horton, H. L. and Ryffel, H. H.: *Machinery's Handbook 28th Edition*. Industrial Press, NY, 2008.

Okuda, T., Hiroyasu, T., Miki, M., and Watanabe, S., DCMOGA: Distributed cooperation model of multi-objective genetic algorithm. In *Advances in Nature-Inspired Computation: The PPSN VII Workshops* (pp. 25-26), 2002.

Ostwald, P.F and Munoz, J., *Manufacturing processes and systems*, Wiley, USA, 1997.

Oxford English Dictionary, http://www.oed.com, accessed (20/1/2013)

Peng, W., Zhang, Q. and Li, H., *Comparision between MOEA/D and NSGA-II on the multi-objective travelling salesman problem*, in Multi-Objective Memetic Algorithms, pp. 309-324, Springer Berlin Heidelberg, 2009.

Prügel-Bennett, A, *When a genetic algorithm outperforms hill-climbing*, Theoretical Computer Science, 320(1), pp. 135-153, 2004.

Rogers, A., and Prugel-Bennett, A., *Genetic drift in genetic algorithm selection schemes*, Evolutionary Computation, IEEE Transactions on, 3(4), pp. 298-303, 1999.

Ronald, S., *Duplicate genotypes in a genetic algorithm*, In Evolutionary Computation Proceedings, 1998 IEEE World Congress on Computational Intelligence, pp. 793-798), 1998.

Runarsson, T. P., and Yao, X., *Search biases in constrained evolutionary optimization*, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, *35*(2), pp. 233-243, 2005.

Russell, S.J. and Norvig, P, *Artificial Intelligence: A Modern Approach (2 ed.).* Pearson Education Upper Saddle River, NJ, 2003.

Salehi, M. and Tavakkoli-Moghaddam, R., *Application of genetic algorithm to computer-aided process planning in preliminary and detailed planning*, Eng. Appl. Artif. Intell., 22(8), 1179-1187, 2009.

Santana-Quintero, L. V., Montano, A. A., and Coello, C. A. C., *A review of techniques for handling expensive functions in evolutionary multi-objective optimization*, In Computational Intelligence in Expensive Optimization Problems, pp. 29-59, Springer Heidelberg, Berlin, 2010.

Sardiñas, R.Q., Santana, M.R., and Brindis, A.E., , Engineering Applications of Artificial Intelligence, 19(2), pp. 127-133, 2006.

Sato, S., K. Otori, A. Takizawa, H. Sakai, Y. Ando and H. Kawamura, *Applying genetic algorithms to the optimum design of a concert hall*, Journal of Sound and Vibration, 258(3), pp. 517-526, 2002.

Savic, D. A., and Walters, G. A., Genetic algorithms for least-cost design of water distribution networks. *Journal of Water Resources Planning and Management*, *123*(2), pp. 67-77, 1997.

Schaffer, D.J., *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*, In Proceedings of the 1st International Conference on Genetic Algorithms, pp. 93-100, 1985.

Segura, C., Coello, C. A. C., Miranda, G., & León, C., *Using multi-objective evolutionary algorithms for single-objective optimization*, *4OR*, *11*(3), 201-228, 2013.

Sengoku, H., and Yoshihara, I., *A fast TSP solver using GA on JAVA*, In *Third International Symposium on Artificial Life, and Robotics (AROB III'98)*, pp. 283-288, 1998.

Sindhya, K., Deb, K., and Miettinen, K., *Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm*, Natural Computing, 10(4), pp. 1407-1430, 2011.

Smid, P., *CNC Programming Handbook: A Comprehensive Guide to Practical CNC Programming (2nd edition)*, Industrial Press Inc., New York, 2003.

Smith, K. I., Everson, R. M., and Fieldsend, J. E., *Dominance measures for multi-objective simulated annealing*. In IEEE *Congress on Evolutionary Computation, 2004 (CEC2004),* pp. 23-30, 2004.

Smith, R. E., Dike, B. A., and Stegmann, S. A., *Fitness inheritance in genetic algorithms*, In Proceedings of the 1995 ACM symposium on Applied computing, pp. 345-350, 1995.

Solimanpur, M., and Ranjdoostfard, F., *Optimisation of cutting parameters using a multi-objective genetic algorithm*, International Journal of Production Research, *47*(21), pp. 6019-6036, 2009.

Spanoudakis, P., Tsourveloudis, N. and Nikolos, I., *Optimal Selection of Tools for Rough Machining of Sculptured Surfaces*, In Proceedings of the international multiConference of engineers and computer scientists Vol. 2, pp. 1697-1702, 2008.

Srinivas, N. and Deb, K. *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*, Evolutionary Computation, 2(3), pp. 221–248, 1994.

T.P.Runarsson and X.Yao. Search biases in constrained evolutionary optimization. IEEE Transactions on Systems, Man and Cybernetics- Part C, 35(2), 233–243, 2005.

Talbi, E. G., Mostaghim, S., Okabe, T., Ishibuchi, H., Rudolph, G., and Coello Coello, C., *Parallel approaches for multiobjective optimization*. In Multiobjective Optimization: Interactive and Evolutionary Approaches, pp. 349-372, Springer Berlin, Heidelberg, 2008.

Thiele, L., Miettinen, K., Korhonen, P.J. and Luque, J.M, *A preference-based evolutionary algorithm for multi-objective optimization*, Evolutionary Computation, 17(3), pp. 411-436, 2009.

Toscano, G. P., and Coello, C. A. C., *The micro genetic algorithm 2: Towards online adaptation in evolutionary multiobjective optimization*, In Evolutionary Multi-criterion Optimization (EMO 2003), pp. 252-266, 2003.

U.S. Congress, *Computerized Manufacturing Automation: Employment, Education, and the Workplace* (Washington, D. C.: U.S. Congress, Office of Technology Assessment, OTACIT-235, April 1984). Available online at http://www.fas.org/ota/reports/8408.pdf

Van Veldhuizen, D. A., and Lamont, G. B., *On measuring multiobjective evolutionary algorithm performance*, IEEE Congress on Evolutionary Computation, 1, pp. 204-211, 2000.

Van Veldhuizen, D. A., Zydallis, J. B., and Lamont, G. B., *Considerations in engineering parallel multiobjective evolutionary algorithms*. IEEE Transactions on Evolutionary Computation, 7(2), pp. 144-173, 2003.

Venkata Rao, R. and Pawar, P. J., *Parameter optimization of a multi-pass milling process using non-traditional optimization algorithms*. Appl. Soft Comput. 10(2), pp. 445-456, 2010.

Vero Software, Machining Strategist 13, Gloucestershire, UK, 2012.

Vosniakos, G.-C. and Krimpenis, A., *Optimisation of Multiple Tool CNC Rough Machining of a Hemisphere as a Genetic Algorithm Paradigm Application*, The International Journal of Advanced Manufacturing Technology, 20(10), pp.727-734, 2002.

Voß, T., Hansen, N., and Igel, C., *Improved step size adaptation for the MO-CMA-ES*, *In Proceedings of the 12th annual conference on Genetic and evolutionary computation(GECCO'10)*, pp. 487-494, 2010.

Voutchkov, I. and Keane, A. J., *Multiobjective optimization using surrogates*, Proceedings of International Conference on Adaptive Computing in Design and Manufacture*,* Bristol, U.K., pp. 167–175, 2006.

Wang, R., Purshouse, R.C., Fleming, P., "Preference-inspired Co-evolutionary Algorithms for Many-objective Optimisation," *Evolutionary Computation, IEEE Transactions on*, 17(4), 2013

Wang, X., and Jawahir, I. S., *Optimization of multi-pass turning operations using genetic algorithms for the selection of cutting conditions and cutting tools with tool-wear effect*, International journal of production research, 43(17), pp. 3543-3559, 2005.

Wang, Y., Ma, H. J., Gao, C. H., Xu, H. G., and Zhou, X. H., *A computer aided tool selection system for 3D die/mould-cavity NC machining using both a heuristic and analytical approach*, International Journal of Computer Integrated Manufacturing, *18*(8), 686-701, 2005(a).

Wang, Z. G., Rahman, M., Wong, Y. S., and Sun, J., *Optimization of multi-pass milling using parallel genetic algorithm and parallel genetic simulated annealing*, International Journal of Machine Tools and Manufacture, 45(15), pp. 1726-1734, 2005b.

Wang, Z. G., Wong, Y. S., Rahman, M., and Sun, J., *Multi-objective optimization of high-speed milling with parallel genetic simulated annealing*, The International Journal of Advanced Manufacturing Technology, 31(3), pp. 209-218, 2006.

Watanabe, K., and Hashem, M. M. A., *Evolutionary computations: new algorithms and their applications to evolutionary robots*, Springer Heidelberg, Berlin, 2004.

Watanabe, S., Sakakibara K., Multi-*Objective Approaches in a Single-Objective Optimization Environment*, Proc. of 2005 Congress on Evolutionary Computation, pp. 1714-1721, 2005.

Whitley, D., Rana, S., and Heckendorn, R. B., *The island model genetic algorithm: On separability, population size and convergence*, Journal of Computing and Information Technology, *7*, pp. 33-48, 1999.

Wickramasinghe, U.K. and Li, X, *Using a distance metric to guide pso algorithms for many-objective optimization.* In Proc. 11th Annual conference on Genetic and evolutionary computation (GECCO'09), pp. 667-674, 2009.

Yagoubi, M., and Schoenauer, M., *Asynchronous Master/Slave MOEAs and heterogeneous evaluation costs*, In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO'12), pp. 1007-1014, 2012.

Yagoubi, M., Thobois, L., and Schoenauer, M., *Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs*, In *IEEE Congress on Evolutionary Computation (CEC 2011),* pp. 21-28, 2011.

Yang, S. H., and Natarajan, U., *Multi-objective optimization of cutting parameters in turning process using differential evolution and non-dominated sorting genetic algorithm-II approaches*, The International Journal of Advanced Manufacturing Technology, 49(5), pp. 773-784, 2010.

Zhang, F., Zhang, Y. F., and Nee, A. Y. C., *Using genetic algorithms in process planning for job shop machining*, Evolutionary Computation, IEEE Transactions on, *1*(4), pp. 278-289, 1997.

Zhang, Q. and Li, H. MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition, IEEE Trans. on Evolutionary Computation, 11(6), pp.712-731, 2007.

Zitzler, E. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, PhD thesis, Swiss Federal Institute of Technology (ETH), 1999.

Zitzler, E., and Thiele, L, *Multiobjective optimization using evolutionary algorithms—A comparative case study*, in Parallel Problem Solving from Nature—PPSN V, pp. 292-301, 1998.

Zitzler, E., Deb, K. and Thiele, L, *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*, Evolutionary Computation, 8(2), pp. 173–195, 2000.

Zitzler, E., Laumanns, M. and Thiele, L., Giannakoglou, K., Tsahalis, D., Periaux, J., Papailou, P. and Fogarty, T. *SPEA2: Improving the strength Pareto evolutionary algorithm*, in Proc. EUROGEN 2001*,* Evolutionary Methods for Design, Optimization and Control With Applications to Industrial Problems, 2001.