Metaheuristic Approaches to Tool Selection Optimisation

Alexander W. Churchill
University of Sussex
Department of Informatics
Falmer, Brighton BN1 9QJ
alexanderchurchill@gmail.com

Phil Husbands
University of Sussex
Department of Informatics
Falmer, Brighton BN1 9QJ
philh@sussex.ac.uk

Andrew Philippides
University of Sussex
Department of Informatics
Falmer, Brighton BN1 9QJ
andrewop@sussex.ac.uk

ABSTRACT
In this paper we discuss our approach to solving the tool selection problem, specifically applied to rough machining. A simulation model is used to evaluate tool sequences, which gives accurate values for tool paths and a 3D model of the final surface of a machined part. This allows for a largely unrestricted search using many different tool types, making this approach more useful for real world applications than previous attempts at solving the problem. An exhaustive search of every valid tool sequence is executed and shows that assumptions present in related research can prevent the optimal solution from being found. Metaheuristic algorithms are used to traverse the search space because of its complex combinatorial properties. Four algorithms are tested - Genetic Algorithm (GA), Stochastic Hill Climbing (SHC), Hybrid Genetic Algorithm (HGA) and Random Restart Stochastic Hill Climbing (RRSHC). Evaluating their performance at coping with two competing demands, finding optimal solutions and keeping the number of potentially expensive evaluations low, it is shown that RRSHC performs best in terms of solution accuracy but at the greatest computational cost. SHC finds the optimum sequence less frequently but needs far fewer evaluations and the HGA lies somewhere in between.

Categories and Subject Descriptors: G.1.6 [Numerical Analysis]: Optimization – Global Optimization; I.2.1 [Artificial Intelligence]: Applications and Expert Systems – Industrial Automation; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search – Heuristic Methods; I.6.3 [Computing Methodologies]: Simulation and Modeling – Applications

Keywords: Tool Selection, Milling, Roughing, Memetic Algorithms, Computer Aided Manufacturing (CAM), Genetic Algorithm (GA), Stochastic Hill Climbing

1. INTRODUCTION
The process of milling involves the use of one or more cutting tools to machine solid material. Machine tools come in a plethora of shapes and sizes and operate at a variety of different speeds. The tool selection problem, which is tackled in this paper, comprises of finding a sequence of tools that will machine a solid to within the desired surface tolerance of a specified profile, in the shortest amount of time possible.

A simulation was used to evaluate each tool sequence, which gave accurate tool paths and a detailed 3D model of the final machined surface. Using this simulation approach means that there are fewer assumptions and restrictions present than in related projects, a selection of which are discussed in section 3. Different shaped tools were permitted in the sequences, rather than restricting the tools to one type and no tool is fixed in the series before the search process takes place. This creates a complex combinatorial
search space well suited to metaheuristic methods. In section 5, four algorithms are evaluated on their ability to deal with two competing demands, finding good solutions and using a small number of potentially expensive evaluations. Modifying the mutation operator to make more conservative changes is demonstrated to be beneficial, whilst it is also shown that hybridising a Genetic Algorithm drastically improves performance.

To the best of the Authors’ knowledge this is the first time that a simulation model has been used which allows for flat endmill, ballnosed and toroidal tools (with the easy addition of others), in a variable length sequence without any tools being fixed in the sequence in advance, making it particularly relevant to industrial use. [insert benchmark/algorithm performance]

2. The Tool Selection Problem in Rough Milling Operations

Machining in Computer Aided Manufacturing (CAM) is often divided into two types of operation, roughing and finishing cuts. This paper takes the same approach as [12] and [6], considering rough milling (or roughing) to be all of the cuts used to progress to the finishing stage. In the literature, briefly reviewed below, roughing is performed using one of two methods. The first technique, seen in figure 1(a), divides the component into a series of vertical slices. A tool is chosen to cut each slice to within a predetermined minimum of excess material (or stock). If the same tool can cut adjacent slices they will be merged into one, otherwise there will need to be a tool change. In the second technique, seen in figure 1(b), a tool is chosen which will remove as much material as is it is able to without gouging the sides of the component. If the tool is too large to remove all the material desired, then a smaller tool will be used afterwards. This process repeats until the sequence of tools has removed all the necessary material.

The second technique is used in this work because it is much more prevalent in industry, as the larger and stronger tools remove as much material as possible, reducing the load on the smaller tools and therefore the risk of costly breakages. In this paper the tool selection problem is confined to roughing only.

3. Background

In industry, tool sequence selection is typically achieved through expert knowledge. However, there have been a number of attempts to automate and optimise selection.

Krimpenis and Vosniakos have taken two different approaches to the problem. An early investigation used a Genetic Algorithm (GA) to select a tool sequence that minimises machining time of a hemispherical shape [16]. The slice by slice roughing method seen in figure 1(a) was used. The GA was able to discover optimal tool sequences after 3000 evaluations. A limitation here was the consideration of flat endmill tool types only. A later work used a different approach, combining two GAs to select a single tool and all of the associated cutting parameters available on a particular CAM software package [11]. The GA was multi-objective, leading to solutions on a three dimensional pareto front. The system was able to find very good solutions for each objective function but using a single tool means it’s unlikely to create an efficient plan in most cases.

Lin and Gian also use the slice by slice technique in their roughing optimisation system [13]. Unlike the work above, tool paths were not used to evaluate solutions. Instead, the largest tools able to machine each slice are selected. Tools operate at different cutting speeds, so looking at geometry alone means that sequences with slower overall machining time may be chosen when using this approach. Another limitation is that only flat endmill tools are considered.

Several tool types are available for use in the work of Wang et al. [17]. Their system used the slice by slice roughing technique and employed expert knowledge for choosing tool types, while taking a similar approach to [13] for selecting tools. However, the tool’s material removal rate (MRR) was included in the selection decision, which accounted for differences in speeds. A shortcoming of this system could be seen in using expert knowledge to choose tool types, as assumptions could prevent optimal tool sequences from being found.

Spanoudakis et al. took a GA approach to optimising rough machining [15]. Their work is similar to [11] in that the GA both selects a tool and optimises associated cutting parameters. A CAM simulation is also used to evaluate solutions. A difference to [11] is that they use the second roughing technique, seen in figure 1(b). Two types of tools are allowed, flat endmill and ballnose. The finishing operation is also included in the optimisation and one tool is selected for each operation. Restricting the selection to one tool per operation is a large limitation as more efficient solutions may be found by combining multiple tools.

A similar approach is found in [1], although only flat endmill tools are used. Importantly, in this work a GA chooses a variable sequence of tools. Good solutions are found at the end of the search. The last tool of the sequence is fixed at the start, which could prevent optimal solutions being found if the work was extended to include different tool types.

Lim et al. [12] use a geometrical approach similar to [17] with the second roughing technique. The largest tool that can remove all the material is chosen as the last tool in the sequence and subsequent tools are chosen based on their MRR. This approach can find good solutions with a low evaluation cost because tool paths are not calculated. However, the way that the last tool is fixed could prevent the optimum sequence from being found. Also, only flat endmill tools are considered.

D’Souza, Sequin and Wright use a graph based method with the second roughing technique [6]. Their search process finds optimal solutions quickly as the geometrical profiles left by flat end mill are taken advantage of to make the search more efficient. However, this can’t be extended to include different tool types.

Bouaziz and Zghal take a similar approach to [12] but use cutting data from an analytical model [3]. They divide what we call roughing into many smaller operations. Their main roughing operation uses an optimised sequence of flat endmill tools which is then followed by an “under roughing” operation which uses a spherical tool. The approach taken in this paper differs from this by not placing restrictions on the stage that different tool types can be used.

Carpenter and Maropoulos [4] and [5] provide a method for optimising tool selection on an array of different milling operation geometries. The user is required to explicitly choose the geometry of the operation, and can’t go as far as a free form sculptured
surface. The system uses a combination of user input and expert knowledge to provide solutions. Whilst being very in depth and allowing for a vast range of operations, it limits the user to only using one tool for each operation where as a sequence of tools may provide a more efficient solution.

The work in this paper builds upon the previous research described above but extends it by allowing for a wider range of tool types and fewer restrictions in tool sequences. The approach taken here is described in detail in the next section.

4. Optimising Tool Selection with Metaheuristic Algorithms

Metaheuristic algorithms are non-problem specific search strategies that use stochastic methods to attempt to efficiently explore a problem space to find (near) optimal solutions [2] and [14]. An investigation was carried out into evaluating how four different metaheuristic search algorithms perform on finding a tool sequence which minimises machining time whilst achieving less than 1mm of excess material when roughing a given geometrical shape. The steering wheel shaped part seen in figure 2 was the main component tested. A library of 18 tools was chosen from a tooling catalogue, which also provided cutting speeds and feeds.1

Sequences are optimised considering only machining time and remaining material, while in the real world there are other important factors to consider such as tool wear. These are ignored here in order to create an initial working model.

4.1 Tool Sequences

A simulation based model was used to evaluate the sequences, as in [11], [15] and [1]. This model allows for the use of different tool types and accommodates future extensions such as including tool holders and tool life. Extending the work of [15], toroidal tools were used in addition to flat endmill and ballnosed cutters. The more industrially relevant roughing technique seen in figure 2(b) is used. Tool sequences were limited to a minimum length of 1 and a maximum of 5 tools. Only the selection of tools and the length of the sequence were optimised. Other cutting parameters were set to defaults controlled in the CAM software.

Without any restrictions on the tool sequences in terms of repetition and order, there are around $40^{12}$ 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 an 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 constraints, inspired by [6], were applied to the tool plans in order to remove this redundancy:

1. Never use an endmill cutter whose diameter is $\geq$ flat diameter of any cutter already used.
2. Never use a ball nosed cutter whose diameter is $\geq$ flat diameter of any ball nosed cutter already used
3. Never use a toroidal cutter whose diameter is $\geq$ flat diameter of any ball nosed cutter already used.
4. Never use a toroidal cutter whose flat diameter $\geq$ flat diameter of any end mill cutter already used.

Using these rules, the total number of sequences was reduced to 39607. Four metaheuristic algorithms were applied to this search space, Stochastic Hill Climbing (SHC), Random Restart Stochastic Hill Climbing (RRSHC), Genetic Algorithm (GA) and a Hybrid Genetic Algorithm (HGA).

4.2 Representation

All of the algorithms represented solutions in the same manner. Tool plans were variable in length but restricted to a maximum of five tools. This meant that sequences could be any length between one and five tools. To allow for this variable length, the tool plan was represented as a string of five entries, with an option for a 'junk' value which would be ignored when read. The tool plan is rendered as each non-junk entry, and its order set by its position in the string, with the first tool on the left. This is shown in figure 3.

![Figure 2: A steering-wheel shaped component that tool sequences were optimised to rough.](image)

1 Tools and their recommended speeds and feeds were taken from Industrial Tool Ltd’s catalogue found at http://www.itc-ltd.co.uk
4.3 Evaluation
To evaluate sequences Vero Software’s Machining Strategist CAM software was used, which simulated the whole tool sequence giving a full path for each individual tool and a 3d mesh representation of the final component. From this, the maximum height of excess stock and the combined machining time of the sequence can be obtained.

For each tool in the sequence, the length of the tool paths were divided by the relevant cutting speeds to give a time per tool, \( t_r \). The goal of the search is to find the fastest tool sequence which will cut a component to a maximum surface tolerance of 1mm. If the tool sequence was within this surface tolerance, then only the overall time for the sequence was used as the evaluation criteria for each solution in the search. If the maximum remaining stock was over 1mm, a time penalty was added to the overall time, using the following method:

\[
\text{If remaining stock} > 1.5\text{mm:} \\
\text{add} \ 2x \text{minutes to the final time} \\
\text{Else if remaining stock} \leq 1.5\text{mm} \geq 1.0\text{mm:} \\
\text{add} \ x \text{minutes to the final time}
\]

The value of \( x \) is determined by calculating the time taken to mill the part using only the smallest tool in the library. A tool change time, \( t_c \), of 30 seconds per change was also added to the total. The simulation time depended on the number of tools in the sequence and the size of the part. Running on a PC with an Intel Core i7 1.6ghz processor, simulations for the part seen in figure 2 ranged from around 10 to 100 seconds. Simulations were cached, so in a run an evaluation was only counted if the sequence had not been seen before. For sequences containing \( n \) tools, the fitness function can be expressed as:

\[
F = x + \sum_{i=0}^{n} t_i + t_c
\]

4.4 Stochastic Hill Climbing
The Hill Climbing search algorithm works by evaluating all of the mutations to an initial solution and choosing the neighbour with the largest improvement. This is repeated until no improvements can be found. SHC is a variant of this where any improvement is accepted, allowing for a less exhaustive search of the neighbourhood [10]. Here, the algorithm was implemented in the following way:

1. Choose a starting tool sequence randomly
2. Mutate the sequence
3. If the mutated plan is a legal combination, evaluate it
4. If the mutated plan has a lower score than its parent, go to 2 using the child plan. Set the number of unsuccessful mutations, \( m \), to 0. Otherwise, go to 2 and increase \( m \) by 1. If this number is larger than 1000, go to 5
5. End and return the lowest scoring tool plan found

4.5 Random Restart Stochastic Hill Climbing
Random Restart is another variant of Hill Climbing. Here, Hill Climbing is repeated from a random starting point after a stopping condition is met [8]. Another stopping condition is used to end the search.

In order to severely restrict the number of evaluations needed by the algorithms, these stopping conditions were carefully planned and tested. To keep an individual hill climbing run short, two conditions were applied:

1. Stop if more than \( n \) evaluations are used, where \( n \) is predetermined
2. If more than 6 iterations have taken place, stop if more than 15 evaluations occur during an iteration

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

\[
\text{If the final member of an individual search run,} \\
B_{mi} \text{ has a fitness higher than or equal to} p \text{ best members of a previous search run then stop, where} p \text{ is predetermined.}
\]

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

1. Choose a starting position for \( x \)
2. Adjust \( x \) using a mutation method
3. Evaluate \( f(x) \) for this new value
4. If \( f(x) \) is lower than when using the previous value for \( x \), set \( x \) to this new value. If we haven’t reached the stopping conditions for an individual search go to 2. Otherwise go to 6
5. If \( f(x) \) is higher than when using the previous value for \( x \) then go to 2. If all neighbouring values of \( x \) have been visited and no lower value found go to 6
6. Add \( x \) to \( B \), a list of the best members. If the stopping conditions for the whole search haven’t been met go to 1, otherwise go to 7
7. Sort \( B \) and return the highest ranking solution

There are two parameters that are set at the start of the search, \( p \) and \( n \). These greatly affect the number of evaluations. To find the best values, the algorithm was run for 100 trials on a number of different parameter values. There was a clear trade-off between the number of times the optimum result was found and the evaluations used. For the comparisons in this paper, the parameters \( p = 2 \) and \( n = 120 \) were chosen.

---

\(^2\) The individual tool path was itself divided into different parts which were operated on at different speeds.
4.6 Genetic Algorithm

Genetic Algorithms use the principles of natural selection to solve problems. They are able to explore all regions of the search space whilst concentrating on promising regions [7].

The GA used here follows this procedure:

1. Create an initial population of \( p \) individuals, randomly chosen from the entire search space. The population size, \( p \), is predetermined.
2. Evaluate each member of the population
3. Rank each member of the population based on fitness
4. Start a new population with the highest ranked member included from the start
5. Have a probability, \( c \), chance of crossover to create a new child. Otherwise, choose a child from the previous population, with a weighted probability based on rank
6. Have a probability, \( m \), chance of mutation applied to the child created in 5
7. Add the child to the new population
8. Repeat 5 to 7 until the population size equals \( p \)
9. Repeat 2 to 8 until the highest ranked member has been the same for \( e \) consecutive generations, where the number \( e \) is chosen in advance.

The GA uses a number of properties to encourage convergence on optimal results and preservation of good solutions. Elitism is incorporated, where the best solution from the previous generation is always included, untouched, in the next generation. This makes sure that we never lose the best tool sequence found. Roulette wheel selection is also utilised, where members are selected from the previous generation with a probability proportional to their fitness. This means that we are more likely to include the better members of the previous population in the new population. However, lower ranked members still have a chance of making it to the new generation, which keeps diversity in the population with the goal of avoiding getting stuck in areas of local optimality.

The crossover operator works by first choosing two parents using roulette wheel selection. The crossover point and the crossover size are chosen randomly. Genes are swapped at the crossover point for the length of the crossover size, creating two new children. These children are then checked to see if they are valid tool sequences. If both children are, then one is randomly chosen. If only one child is valid, it is used in the new population and if neither are legal then a child is created using the normal roulette wheel selection technique.

As with RRSHC, there are a number of parameters present in the algorithm that need to be set before search takes place. These are \( p, c, m \) and \( e \). The stopping condition \( e \) was fixed at 25 generations without improvement. Within predefined constraints, all the other parameter combinations were tested for 100 trials each. It was again seen that the frequency that the optimum solution was found correlated with a higher number of evaluations. The parameters chosen for comparison were \( p = 9, m = 0.4, c = 0.7 \).

<table>
<thead>
<tr>
<th>Tool A</th>
<th>Tool B</th>
<th>Tool C</th>
<th>Tool D</th>
<th>Junk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tool A, Tool B, Tool C, Tool D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The GA used here follow...
Once again combinations of parameters were tested for 100 trials. The parameters chosen for comparison were \( n = 3, \ i = 6, \ e = 30 \) and flat mutation on.

### 4.8 The Mutation Operator

All of the search procedures use the same mutation operator. Mutations are carried out by changing one of the five entries in the string to either one of the eighteen available tools or to the ‘junk’ 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. In the Stochastic Hill Climbing algorithm, if a valid plan can’t be found after 1000 mutations the search will end. Every valid tool sequence one mutation away from the original is considered to be a neighbour of the original. The mutation process can be seen in the figure 4.

### 5. Results

#### 5.1 The Search Space

An exhaustive search was carried out on the search space, meaning that every valid tool sequence combination was simulated. This was done to find the optimum solution, to analyse the validity of assumptions made in previous research projects and to map the search space in order to effectively judge the performance of the metaheuristic algorithms.

Figure 5 shows that over a third of the valid tool sequences take between 200 and 300 minutes to complete. Only 618 solutions reside in the 100 – 200 minute band. Figure 6 shows a plot of the machining time taken by the 100 best tool sequences.

A significant result from this search is that the best result, which takes around 116 minutes, uses a 12mm diameter endmill followed by a 6mm toroidal cutter. The second best result uses the same first tool but an 8mm toroidal cutter second. This implies that we would find a suboptimal solution if we used the rules found in [1] and [12], which fix the last tool in sequence to the largest able to reach the desired surface tolerance.

#### Tools are different sizes but also run at different speeds, meaning that while the tool path may be longer with a smaller tool, it could complete its operation sooner if it is able to cut significantly faster than a larger tool. This leads to a complicated combinatorial search space well suited to a metaheuristic search approach.

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

#### 5.2 Evaluating the Performance of the Metaheuristic Algorithms

The search algorithms described in section 4 were each run for 10,000 trials on the component seen in figure 2. The goal of the search was to find sequences with the shortest machining times using as few evaluations as possible. Evaluating using a simulator can be very costly, so it is desirable to keep the total low. Table 1 below shows the results for all of the metaheuristic algorithms described in section 4. All of the search methods could be considered successful in tackling this task. Each of them achieved a median best score equal to the optimum solution and
was 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 algorithms.

There is no clear winner, an algorithm which finds the best solution with the fewest evaluations. It can clearly be seen that hybridising the GA has a very positive effect and improves on the GA search at every level. The frequency that the optimum solution is found is raised by 42% while the 95th percentile of evaluations is 26 fewer.

SHC search is the fastest, needing fewer evaluations than the other algorithms. However, the algorithm is more prone to getting stuck in suboptimal regions of the search space compared to the others. This can be seen in the worse score of 246 minutes which is considerably less than the GA’s 195.64 minutes.

RRSHC is by far the most successful in terms of finding the optimum solution, occurring around 92% of the time, almost 12% more than HGA. However, median and 95th percentile evaluations are more than double that needed by the HGA. The 95th percentile scores are similar for both algorithms.

5.3 Gradual Mutation

5.3.1 Implementation

The mutation operator used by all of the algorithms in the section above could be quite disruptive. 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 can have a big effect on a sequence. There is also a much smaller chance of a tool being removed or inserted than being changed.

The mutation operator was modified to evaluate whether avoiding these problems would make the search more efficient. The new mutation process firstly gave a 0.3 chance of inserting or removing a tool and a 0.4 chance of changing an existing tool without modifying the length of the sequence. When tools were inserted or changed, there was a weighting system which made sure that there was a greater probability of the new tool being close in size to the original. In this way the mutations became less disruptive and the changes more gradual.

5.3.2 Results

Table 2 below shows that using gradual mutation improved the performance of all of the algorithms both in terms of the quality of solution and the number of evaluations needed. However, the relative differences between the algorithms remained the same for the most part. For example, RRSHC and the HGA still reached the optimum more often and SHC still had the fastest search.

The most dramatic differences were 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 was that the worst solution found was better for all the algorithms apart from HGA, where it remained the same. This was most improved in SHC search, which meant that it now outperformed the GA for all criteria.

6. Discussion and Future Work

The results from section 5 show four important outcomes from this work. The first is that the assumption found in some previous work that ending with the largest tool that’s able to remove all the required material is incorrect because it doesn’t take cutting speeds into account. Metaheuristic algorithms are found to work very well on this problem, and a comprehensive comparison in the performance between different varieties is given for the first time, to the best of the Authors’ knowledge. Using a carefully considered mutation operator that makes more gradual changes is shown to significantly improve search performance. Finally, hybridising the GA is also shown to greatly increase the effectiveness of the algorithm.

There is no clear winner among the metaheuristic algorithms, in terms of satisfying the competing demands of finding (near) optimal solutions and keeping evaluation numbers low. RRSHC, found the optimum solution most often and SHC used the lowest median evaluations. SHC performs better than the GA because it finds the optimum more frequently with fewer evaluations. The HGA could be considered to perform better than SHC because 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 doesn’t 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 aren’t as expensive. In this situation the evaluations criteria becomes less important and RRSHC may be seen as 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.

The outcome of this is that there isn’t a single algorithm that should be used in every situation, one must choose the search technique which best suits the requirements. For instance, if the requirement is to find the best possible solution, i.e. the fastest to manufacture, above all else then RRSHC should be used. It can also be chosen if the part is small and the simulations are fast. If the part is very large and it doesn't matter if it takes longer to manufacture then normal SHC can be used. For a better solution that uses a reasonable amount of evaluations, the HGA could be chosen.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SHC</th>
<th>RRSHC</th>
<th>GA</th>
<th>HGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Score (mins)</td>
<td>119.96</td>
<td>116.23</td>
<td>120.6</td>
<td>118.40</td>
</tr>
<tr>
<td>95th Percentile Score</td>
<td>134.73</td>
<td>115.99</td>
<td>134.73</td>
<td>134.73</td>
</tr>
<tr>
<td>Median Evaluations</td>
<td>68</td>
<td>141</td>
<td>94</td>
<td>87</td>
</tr>
<tr>
<td>95th Percentile Evaluations</td>
<td>101</td>
<td>207</td>
<td>139</td>
<td>122</td>
</tr>
<tr>
<td>Best Score (mins)</td>
<td>115.99</td>
<td>115.99</td>
<td>115.99</td>
<td>115.99</td>
</tr>
<tr>
<td>Worst Score (mins)</td>
<td>178.35</td>
<td>134.73</td>
<td>183.63</td>
<td>178.35</td>
</tr>
<tr>
<td>Number of times the optimum solution was found</td>
<td>7942</td>
<td>9853</td>
<td>7834</td>
<td>8761</td>
</tr>
</tbody>
</table>
There are a number of possible extensions to the current work which are currently being investigated. Changing the mutation operator from making a completely random change to a tool sequence to a more considered one had a large and positive impact on the search performance of all the algorithms. An obvious extension to this work would be to attempt a similarly considered method for crossover in the GA based algorithms.

Another way in which search could be improved is by including learning. This could be used in choosing an initial solution or population for the start of search or for choosing mutations or crossover genes in a more intelligent manner based on an analysis of the best solutions from an array of different parts and tool libraries.

Including tool wear and tool cost in the model would make the solutions more relevant and useful to industry. Tools have a limited life and are expensive. There can be a trade-off between how quickly a part can be machined and how much it costs in terms of tool wear. This makes the tool selection problem multi-objective. This extension is currently being investigated.

7. Conclusion
This paper has introduced a new approach to the tool selection problem. A detailed simulation is used to evaluate tool sequences, giving an accurate representation of tool paths and the surface left after the component has been machined. This allows different tool types to be employed and a free search of tool sequences without imposing assumptions, such as deciding what the last tool should be in advance.

Four metaheuristic algorithms were tested on the problem. Each of them was able to find good solutions but there was no clear winner, one which always found the optimal solution in a small number of evaluations. Modifying the mutation operator to make more considered changes made the search more efficient but didn’t help this issue. This means that using this system, it will be up to the user to decided what the requirements are for the search in terms of how good the solution needs to be and how long it is allowed to take. Once these have been ascertained the best suited algorithm can be selected.

8. AKNOWLEDGMENTS
Special thanks to Steve Youngs who provided invaluable help in setting up simulations and also to Gerry O'Driscoll, John Cockerill and Adrian Thompson. This work was supported by Vero Software. Alexander Churchill is supported by an EPSRC Industrial CASE studentship.

9. REFERENCES