CNC Milling Toolpath Generation

Using Genetic Algorithms

CNC Milling Toolpath Generation Using Genetic Algorithms

Wesley P. Essink

A thesis submitted for the degree of Doctor of Philosophy

University of Bath

Department of Mechanical Engineering

September 2016

Signed on behalf of the Faculty/School of .....

Signature of Author .....

Wesley P. Essink

Abstract

The prevalence of digital manufacturing in creating increasingly complex products with small batch sizes, requires effective methods for production process planning. Toolpath generation is one of the challenges for manufacturing technologies that function based on the controlled movement of an end effector against a workpiece. The current approaches for determining suitable tool paths are highly dependent on machine structure, manufacturing technology and product geometry. This dependence can be very expensive in a volatile production environment where the products and the resources change quickly. In this research, a novel approach for the flexible generation of toolpaths using a mathematical formulation of the desired objective is proposed. The approach, based on optimisation techniques, is developed by discretising the product space into a number of grid points and determining the optimal sequence of the tool tip visiting these points. To demonstrate the effectiveness of the approach, the context of milling machining has been chosen and a genetic algorithm has been developed to solve the optimisation problem. The results show that with meta heuristic methods, flexible tool paths can indeed be generated for industrially relevant parts using existing computational power. Future computing platforms, including quantum computers, could extend the applicability of the proposed approach to much more complex domains for instantaneous optimisation of the detailed manufacturing process plan.

Acknowledgements

This research would not have been possible without the continued support from family, friends and colleagues. I would like to take this opportunity to thank all of the people involved in supporting me throughout my postgraduate studies.

Firstly, I would like to thank my supervisor, Dr. Aydin Nassehi for not only the opportunity to perform this research but also for his unending moral, technical and academic support throughout my studies. I will be forever grateful for your continued friendship and patience through even the most difficult moments over the last years.

Secondly, I would like to extend my gratitude to my second supervisor, Professor Stephen T. Newman. His support has been invaluable throughout my postgraduate studies. His sustained encouragement and enthusiasm helped push me to the end of my thesis.

I would also like to thank all of the past and present members of the AMPS research group at the University of Bath who's friendships created a great environment to inspire creative ideas and provide helpful feedback for my research. A special mention to Dr. Joseph Flynn, I will be forever indebted to your friendship and moral support.

Finally I would like to express my utmost gratitude and love to my parents, Gerard and Nicoline Essink. Without your love, support and encouragement none of my studies would have been possible. Thank you.

Contents

List of Figures

List of Tables

Glossary

2D	**2** **D**imensional
3D	**3** **D**imensional
B-rep	**B**oundary-**rep**resentation
ACO	**A**nt **C**olony **O**ptimisation
CAD	**C**omputer **A**ided **D**esign
CAM	**C**omputer **A**ided **M**anufacturing
CMM	**C**oordinate **M**easurement **M**achine
CNC	**C**omputer **N**umerical **C**ontrol
CSG	**C**onstructive **S**olid **G**eometry
C-space	**C**onfiguration **s**pace
EA	**E**volutionary **A**lgorithm
ERX	**E**dge **R**ecombination **Cross**over
GA	**G**enetic **A**lgorithm
IDEF	**I**ntegrated **Def**inition
ISO	**I**nternational **O**rganisation for **S**tandardisation
LK	**L**in-**K**ernighan
LKH	**L**in-**K**ernighan-**H**elsgaun
NC	**N**umerical **C**control
OX	**O**rder **Cross**over
PMX	**P**artially **M**apped **Cross**over
STEP	**St**andard for the **E**xchange of **P**roduct model data
TSP	**T**ravelling **S**alesman **P**roblem

Chapter 1

Introduction

1.1 Background

As the manufacturing industry is becoming increasingly automated with the industry's move from craft production to mass customisation, several paradigm shifts have become necessary[1] from the use of machines to replace hand-made parts, through the use of production lines to increase the volume of products being made, culminating in the increased use of IT in computer integrated manufacturing.

Today, the main challenge is to adapt to the fast pace of technology development and making sure manufacturing technology is sufficiently flexible to accommodate these changes. One of the main issues that need to be tackled is the time taken between design and manufacture[2]. Machining is amongst the core technologies used in the manufacturing process of various products, either to make the product directly or to manufacture enablers such as moulds. Computer Numerically Controlled (CNC) metal cutting machines are utilised extensively in the manufacturing industry. For these machines, with every design change, a new CNC machining program needs to be developed to manufacture the part. This can be very time consuming to do manually[3, 4].

CNC machines are generally programmed by specifying the controlled motion of the various axes of movement that connect the mechanical elements of the machine tool to result in the controlled movement of a cutting tool against a workpiece to cut away the excess material to produce the desired shape of the end product. These axes movements can also

be seen as relative movement of the cutting tool against a stationary workpiece and many researchers consider such "toolpaths" as the main frame of reference in programming CNC machine tools.

The current method of generating toolpaths requires significant input from an expert user, the quality of the generated paths is highly sensitive to the geometry of the part and distinct algorithms are required to obtain paths that optimise certain characteristics of machining processes.

This research aims to specify and realise a novel methodology of generating toolpaths in a flexible manner so that various optimisations are possible independent of the part geometry, without requiring a database of different algorithms.

Evolutionary optimisation in general, and genetic algorithms in particular, underpins the theoretical framework of the proposed methodology.

Genetic algorithms can easily be adapted for optimisation of additional objectives which makes them an excellent platform for toolpath generation when various characteristics are given importance in the machining process.

The following section will cover the framework of the research which is followed by a review of the current methods for generating toolpaths in Chapter 2. Chapter 3 will provide a theoretical framework for the models and concepts used to implement a prototype solution to the research problem. The development of this prototype is discussed in Chapter 4. This is followed by a set of test cases designed to validate the developed prototype in Chapter 5 with the results of these tests cases provided in Chapter 6. A discussion of the research performed is given in Chapter 7. Finally the conclusions of the research along with the author's vision of potential future work are provided in Chapter 8. An outline of the thesis structure can be seen in Figure 1-1.

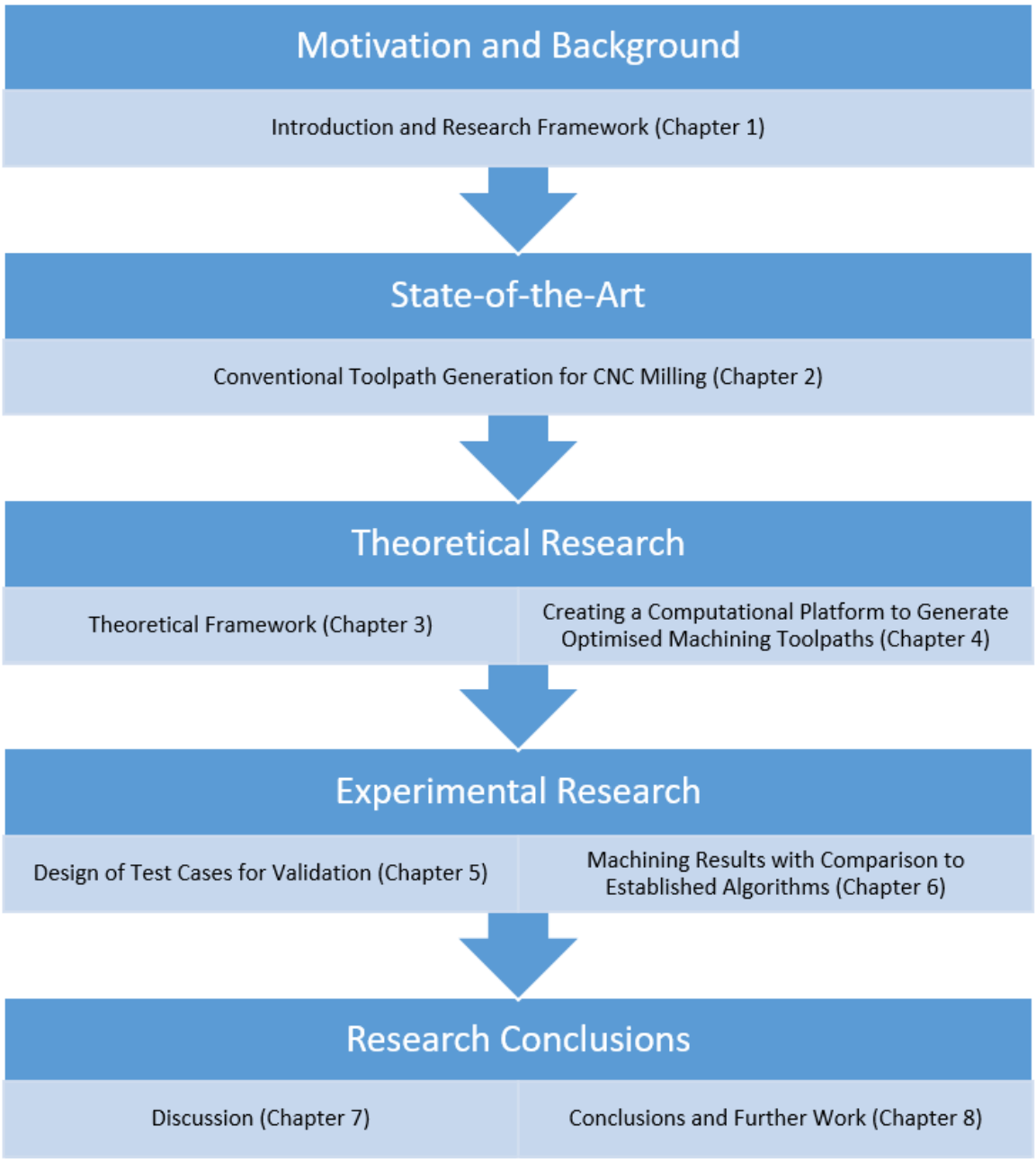


Figure 1-1: Structure of Thesis Chapters.

1.2 Research Aim

This research was conceived due to an industrial need to increase interoperability in the manufacturing chain from design to finished part. It was found that the best way to increase interoperability was to increase the manufacturing information passing between the various stages in the manufacturing chain[5]. Currently the information chain is very much unidirectional with some information being lost along the chain. An example of this can be identified by considering a part designed in CAD which contains all of the geometry and tolerances for the part. At the CNC machining stage, all the information that is left is the G code which describes the axial movements of the tool and the machining parameters for the CNC machine.

To overcome this loss of information, a new machine tool control language was developed called STEP-NC[6]. This new control language would contain all of the manufacturing information required at any stage of the manufacturing process (part geometry/tolerances, manufacturing operations etc.). From the literature in Chapter 2 it can be seen that there is still a lack of available software and tools to properly utilise this new control language. STEP-NC has still not been adopted by industry which means that current CNC machining is still done using G Code.

Therefore the overall aim of this research is to develop a flexible computational platform that can generate efficient CNC milling machining toolpaths for parts described by the STEP-NC data structure. This will allow the STEP-NC representation of a part to be used on CNC machines using the G Code programming language. The computational platform developed in this research will replace the CAM stage in the manufacturing chain thus it will have to be flexible enough to handle various part geometries and features as well as milling tools and machining strategies.

1.3 Research Objectives

The following set of objectives were defined to create a structure for the research and to ensure the aim was achieved:

- Identify and convert part geometry from STEP-NC to programming environment.
- Adapt travelling salesman problem model to machining toolpaths
- Analyse part geometry and model features into separate travelling salesman problems.
- Develop algorithm to solve the travelling salesman problem.
- Adapt algorithm to generate machining toolpaths to be used on a CNC milling machine.
- Validate the developed algorithm by comparing to currently used tools to generate CNC milling toolpaths.

1.4 Research Boundaries

This section will identify the various boundaries of this research and define what lies within the boundaries and what does not. A summary of the research boundaries can be seen in Figure 1-2

1.4.1 CNC Machining Toolpaths

Generating machining toolpaths is an important part of the manufacturing process as it describes how a tool will move around a part to remove material and create the finished product. There is a wide variety of CNC machines used in industry which all require machining toolpaths. Milling machines, lathes, wire EDM and additive machines are all examples of machines which utilise toolpaths. However the way in which the toolpaths for these various machines are generated from features can be fundamentally different and therefore it was decided for this research to focus on just looking at milling machines.

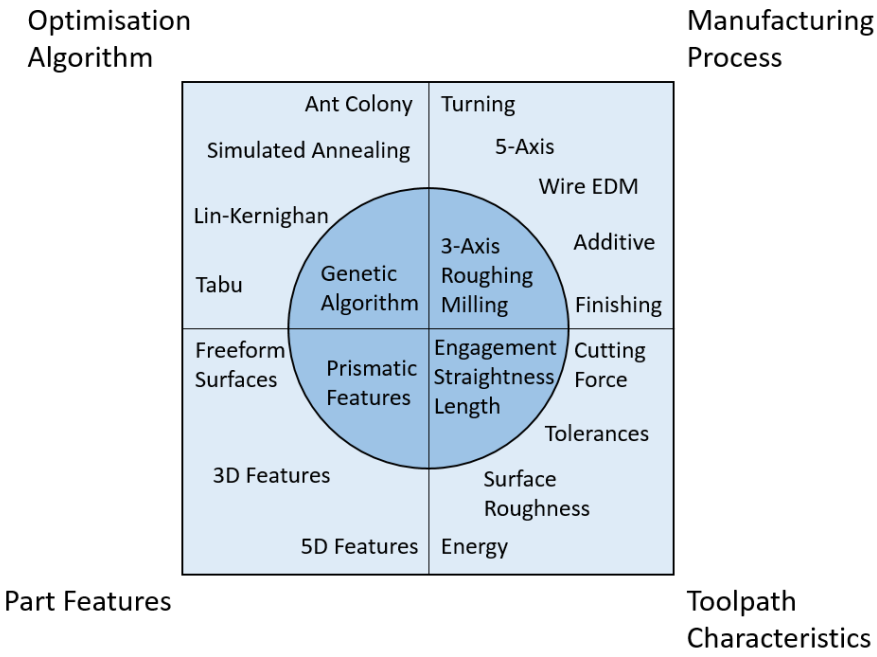


Figure 1-2: The research boundaries of this research.

Milling machines can have a number of different kinematic set-ups which will alter the complexity of the toolpath to be used on each type of milling machine. Milling machines can have up to three linear axes and three rotary axes. The majority of milling machines in industry will either be 3-axis (three linear axes) or 5-axis (three linear and two rotary axes). From the literature in Chapter 3 it can be seen that the travelling salesman problem has only been modelled as a two or three dimensional problem. Therefore the scope of this research will limit the toolpath generation to 3-axis milling machines.

Finally with regards to machining toolpaths, there are two types of machining that is typically done. There is the roughing machining process which removes the bulk of the material at a relatively high rate. This is then proceeded by the finishing machining process which removes the final layer of material at a lower rate to ensure a certain level of surface finish and accuracy. This research will focus on the roughing process as the toolpaths are longer and form a larger part of the machining process, therefore there is more potential for optimisation of these toolpaths.

1.4.2 Part Geometries and Features

The potential combination of features and geometry on a part is virtually infinite. However, this research will focus primarily on prismatic parts but will also include prismatic features that have sloped boundaries to test the toolpath generating algorithm on some basic 3D features.

1.4.3 Optimisation Algorithms

As identified by the literature review performed in Chapter 3.6.2, quite a few meta-heuristics have been developed with the aim of solving the standard travelling salesman problem. This research aims to develop a genetic algorithm to attempt to solve a modified version of the travelling salesman problem and generate machining toolpaths. The reason for choosing genetic algorithms is for their modular structure and their flexibility in switching between individual and multiple objective functions.

1.5 Scope of the Research

This section will define the research scope which has been identified to achieve the research objectives outlined in Section 1.3:

1.5.1 Review of State-of-the-Art in Toolpath Generation and Optimisation for Milling Technologies

The current methods of toolpath generation used in industry as well as used in research have been reviewed and assessed in Chapter 2 alongside a review of the different toolpath characteristics that are focused on by optimisation algorithms and the methods of doing so. This literature review identifies the research gaps in this area and provides support for the objectives specified in this chapter.

1.5.2 Review of Current Methods for Solving the Travelling Salesman Problem

The travelling salesman problem is a computationally difficult path optimisation problem. A large number of heuristics, meta-heuristics and algorithms have been developed in the effort of solving the travelling salesman problem. A review of the most commonly used algorithms and heuristics in solving the travelling salesman problem has been performed in Chapter 3 with a focus on evolutionary algorithms and in particular genetic algorithms as these were identified to be the most feasible method of solving an adapted travelling salesman problem with respect to machining toolpaths.

1.5.3 Specification of a Novel Framework for Realisation of Toolpath Generation Using a Genetic Algorithm

A theoretical framework of a system used to model and solve machining toolpath generation as a travelling salesman problem was developed based on the information reviewed in Chapters 2 and 3. The specification and development of the theoretical framework can be seen in Chapter 3.

1.5.4 Modelling Toolpath Generation as a Travelling Salesman Problem

A method of defining machining toolpath generation as a travelling salesman problem is outlined in Chapter 4. This process analyses features defined by the STEP-NC standard and allows for toolpaths to be generated that are defined by the G Code standard. Three machining toolpath characteristics were modelled into the TSP objective function format to allow for optimisation of these characteristics.

1.5.5 Developing a Computational Platform to Solve the Travelling Salesman Problem

The development and functionality of a flexible and modular optimisation algorithm designed to solve an adapted travelling salesman problem is discussed in Chapter 4.3. The optimisation

algorithm was designed to produce legal machining toolpaths to be used on a 3-axis milling machine. The objective functions defined in Chapter 4 are used by the optimisation algorithm to produce optimised toolpaths with respect to these objective functions in the context of a 3-axis milling machine tool.

1.5.6 Evaluation of the Toolpath Generation Computational Platform

The design of two test parts and a series of test cases is outlined in Chapter 5. One test part is used as a validation tool of the computational platform's abilities with respect to an industrially inspired part. It is also used to measure the performance of the optimisation algorithm which can be compared to other commonly used computational platforms. The other test part and test cases are used to analyse the performance of the optimisation algorithm as well as validating the models and methods developed in this research.

Chapter 2

Toolpath Generation for CNC Milling

2.1 Model Representation

The first stage of generating a machining toolpath is to have an accurate representation of the part you are going to manufacture. It is important to have a sufficiently accurate model of the part to ensure that any subsequent methods of toolpath generation will yield useful results that are within tolerance[7].

A simple and easy to implement method of modelling a part is by using constructive solid geometry(CSG)[8]. CSG uses boolean operators to combine simpler shapes into more complex shapes or surfaces. The most common operators in CSG are the difference, intersection and union operations. Geometric transformations can also be performed in the construction process. An example of how a CSG part is created and represented can be seen in Figure 2-1[9]. Although this method is user friendly, it is quite difficult to create very complex surfaces such as free-form surfaces. To create these, a more flexible modelling system is required.

Boundary representation(B-rep) is another method of model representation which allows for a more complex part or surface to be modelled than by using CSG as it is more flexible and has a wider range of operations[10]. B-rep uses many surface elements connected together to

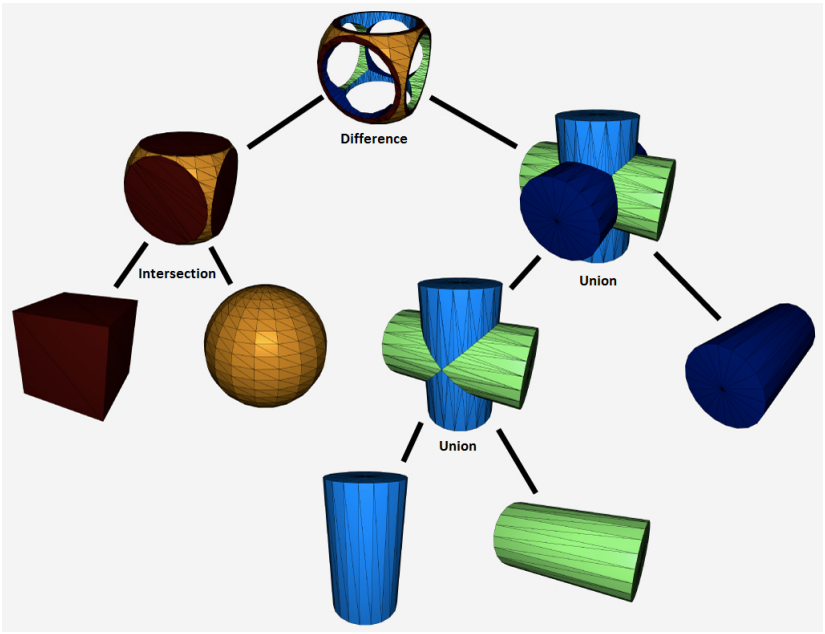


Figure 2-1: The construction of a part represented with CSG.

create a model of the entire surface of a part[11]. Each individual surface can be described by a number of different types of curves such as: Bezier, B-Spline, or NURBS[12]. B-reps can be used in the STEP data structure to store surface information of a part or feature.

A part can also be modelled as a collection of voxels[13]. Depending on the accuracy required, the size of each voxel can be adjusted when creating the model. The benefits of using voxels is that each voxel has a specific coordinate which can be used to perform calculations on the surface however the drawback is that if a high accuracy is required then the number of voxels in the model can be very large and therefore require a lot of memory to store the information. Figure 2-2[14] illustrates how voxels can be used to represent a part.

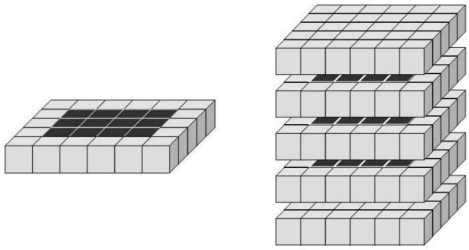


Figure 2-2: Solid model representation using voxel layers.

A surface of a part can also be represented using a point cloud. This can be done by sampling a virtual or real surface in three dimensions and storing the data points in a collection or cloud. The higher the frequency of sampling, the more points that are generated in the point cloud. A number of toolpath generation techniques have been developed using a point cloud method[15, 16].

2.2 Toolpath Generation for Milling Machines

Toolpath generation is an important step in any process that requires precise and controlled movement of an end effector against the product that is being manufacturing. Effective machining, in particular, is reliant on toolpaths that are appropriate for the technology, materials and equipment that are being used. An inappropriate toolpath can lead to part features being machined out of tolerance, low quality surface finish or excessive machining time due to inefficiencies. Generating efficient toolpaths is a major area of research as manufacturing companies can potentially save time and money by incorporating new algorithms into their manufacturing process.

There is a wide variety of literature available on toolpath generation as there are many methods of solving this complex problem. A survey paper was written by Dragomatz and Mann in 1997 which covers many of the widely used methods required to generate toolpaths for both three-axis and five-axis machines as well as a few non-traditional methods[17]. They classified the literature based on a number of keywords such as isoparametric paths (where focus is on surface machining based on parametric spaces); planar pocketing paths (where the focus is on 2D pockets); tool positioning (where the focus is on precise location of the tool); and, roughing paths (with the focus on removing large quantities of material quickly). Whilst efficiency improvements can be made throughout the process, with the large quantities of material removal in rough machining, efficiency gains could be more substantial. Most authors have selected the scope of their research based on technology rather than the process planning stage to focus on either on the generation of toolpaths for three axis milling machines or those for sculptured surface on five axis machines.

2.2.1 Toolpath Generation for Three-Axis Machining

Generally, fewer variables are considered in toolpath generation for three-axis machining compared to five-axis machining as the number of control mechanisms is smaller. Effectively, there are only three variables to consider for each point on the path, linear interpolation between two points results in a linear trajectory of the tool on the machine and the cutter contact point is relatively simple to calculate[18].

Traditional methods of generating toolpaths for three-axis machines involve drawing parallel lines to the contours on the part[19] with the aim of maintaining constant proportion of tool engagement to ensure consistent wear. This can be done in a variety of ways: one method is to use Voronoi diagrams to segment the space within a boundary into sections and then using offset curves to generate the toolpaths in each space[20]. Figure 2-3 shows how the feature is separated into segments using the Voronoi method. By moving the boundary points along the lines of the Voronoi diagram at a constant rate, which can see seen in Figure 2-4, a series of contours parallel to the original feature boundary can be generated.

Figure 2-3: An example of a Voronoi diagram for a simple feature boundary.

Figure 2-4: Offset contours generated from Voronoi diagram.

Persson developed an algorithm which can generate NC machining toolpaths for arbit-

rarily shaped pockets[21]. By segmenting the pocket using bisector lines for each contour of the pocket, scaled contour lines can be drawn into the pocket which can be joined together to form a spiral which follows the contour of the pocket. This method has proven to work well, however can only be used for closed pockets. This method only generated machining toolpaths which follow a single machining strategy and still required user interaction to complete.

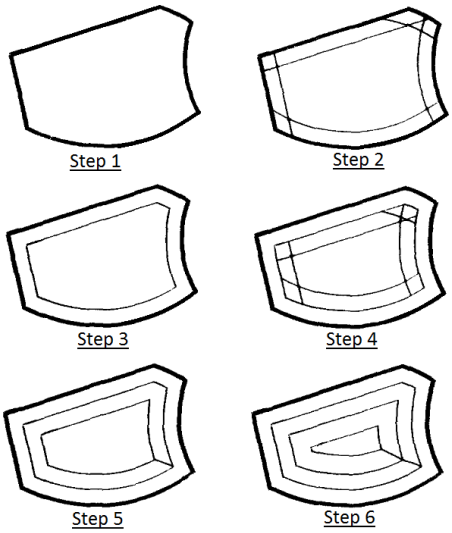


Figure 2-5: Scaled contour lines generated for an example feature using bisector lines

An algorithm which produces a spiral toolpath between a set of points was developed by Qiu[22]. This method uses the minimax approximation and produces very smooth curves from the outermost point to the innermost point. Although the method generates good results, the method is very limited in its application. It only produces curved arcs between points and does not produce full machining toolpaths for features or parts.

Yeung developed an algorithm which produces smooth curves between points in a machining toolpath[23]. This method uses arc-splines between points to smooth the interpolation between the two points. The developed algorithm also reduces the number of data points required to represent the toolpath by finding arcs which pass through as many points as possible. This converts a multipoint linear interpolation to a single arc interpolation. This is a very useful study which does not produce NC machining toolpaths but does optimise

existing NC machining toolpaths.

Arkin performed a study in which a toolpath adhering to a zig-zag machining strategy would be generated with the least number of tool retractions[24]. Toolpaths were only generated for pocket features but yielded good results. However, as discussed in the study, the toolpaths were not optimal as some retractions were more acceptable than others. The toolpaths generated only generated one type of machining strategy for one type of feature which limits the global use of this method for an entire part.

Park investigates a toolpath generation algorithm for bidirectional milling which aims to find the best orientation for the toolpath with respect to the feature being machined[25]. The criteria Park uses to find an optimal orientation is the number of retractions the tool has to perform and the average length of each element of the toolpath. The results of the study show the algorithm performing well and finding the optimal orientation for the test parts.

A STEP-NC compliant CNC milling machine of 2.5D parts was developed by Lee[26]. The information required to machine the various features is located within the STEP-NC data structure and can be used to generate machining toolpaths. The test part produced by the study was had very simple features which indicated the limitations of the developed controller. This research aims to develop a more complex toolpath generation algorithm to increase the complexity of the part that can be produced. This will greatly increase the usefulness of the algorithm as it can be applied to a much wider range of parts.

Generating machining toolpaths for free-form or sculptured surfaces is more challenging as the surfaces of the part are more complex. The toolpath generation methods mentioned above work well for pockets and simple features but fail to generate viable toolpaths for free-form surfaces. The following literature in this section covers toolpath generation for free-form surfaces for three-axis machines.

Radzevich outlines the various conditions that need to be met in order for a sculptured surface to be properly machined[27]. The six conditions which are explained in detail within the paper will need to be taken into consideration when assessing the quality of any machined part with a sculptured surface in this research. The quality of any test parts produced can

be defined by how well the various conditions proposed by Radzevich are met.

A toolpath generating algorithm for free-form surfaces was developed by Bobrow which generates machining toolpaths from CSG part representations[7]. The user must select the various part surfaces to be machined as well as check surfaces. Then the algorithm slices the part surfaces with planar surfaces and produce intersection curves which will define the lines of the machining toolpath. Although this algorithm produces machining toolpaths for full parts, it is not fully automated and does not produce toolpaths which follow a machining strategy.

Elber introduced a new method for generating 3-axis toolpaths for sculptured surfaces[18]. Instead of using regular iso-parametric curves to generate toolpath lines along a surface, an adaptive iso-parametric curve algorithm was developed. This ensured that there was a consistent spacing between curves even for complex geometry where the density of the curves could vary. This resulted in machined surfaces with a better quality surface finish due to a consistent scallop height between each pass.

Choi uses a configuration space approach to generate toolpaths for 3-axis machining of free-form surfaces[28]. By analysing the surface of the part and comparing that to the tool dimensions, a cutter location surface can be generated by identifying the areas the tool can be positioned in without gouging or colliding with the part. A Z-map can be created from this surface with a defined grid spacing depending on the tool size. A toolpath is then generated from the Z-map by following the grid points in a certain direction. This approach yielded good results with non-gouging/colliding toolpaths being generated for a complex free-form surface.

A study performed by Lin develops an algorithm which generates NC cutter toolpaths for sculpted surfaces from massive data point sets[16]. The surface of the part is separated into slices depending on the cut depth. A mesh is then created over the entire surface and a toolpath is generated which follows the contours of the mesh. The disadvantage of this technique is that the generated toolpaths are not optimal and do not follow any machining strategies.

A similar study done by Chui presents an algorithm to generate NC cutter toolpaths for a ball nose cutter from massive point clouds[15]. This is done by separating all of the points into bands whose width is dependant on the scallop height of the cutter. The generated tool paths move through all of the points in each band to create the machined surface. This technique is very limited in the fact that the surface finish is very poor and would only be appropriate for a roughing cut as well as the lack of machining strategy being applied to the machining process. Chui improved this process by converting the point cloud into a 3D triangular mesh and then calculating the local surface normal vectors to the mesh[29]. A toolpath could then be generated by linking the surface normal vectors with bi-arcs.

Lo discusses the issue of generating constant feedrate toolpaths with respect to cutter locations[30]. Lo states that the path between cutting locations is different to the path between cutter contact points, which can lead to a variation in the feedrate between cutter contact points. This can result in a decrease in the surface finish of the machined part. To overcome this issue, Lo developed an algorithm to compensate for the difference between cutter location and cutter contact paths which resulted in an increase in surface quality.

Ding proposes a modified algorithm for the iso-planar toolpath generation method[31]. Surfaces which contain both regions of high surface variability and low surface variability generate redundant toolpaths using the traditional iso-planar toolpath generation method. Therefore to overcome this, areas with high surface variability are identified and an iso-planar toolpath is generated for each individual region. This new method produces much more efficient toolpaths when compared to the traditional iso-planar method.

Yuwen states that the iso-planar has limitations in toolpath generation and therefore proposes an iso-parametric method of generating toolpaths which. First the surface is represented as a triangular mesh and then iso-parametric curves are created which follow the boundary curves of the surface. This method can produce bidirectional or spiral toolpaths depending on the surface geometry but not a combination of both.

Feng introduced a new method of calculating cutter location paths[32]. Instead of using an iso-parametric or iso-planar method, a constant scallop height method was developed. This

method would ensure that the scallop height remained constant across the whole surface and within the specified tolerance. This was achieved by drawing scallop curves iteratively across the free-form surface. The resulting toolpaths from this new method were shorter and more efficient than the iso-planar and iso-parametric methods.

Wu proposed an algorithm to generate machining toolpaths for three-axis free-form surfaces using a 3D Z-map representation of the surface[33]. The algorithm produces unidirectional roughing and finishing toolpaths. The Z-map resolution is adjusted depending on the error tolerance and the tool size. The toolpath is then generated by following the height of the part across the surface for each pass. The drawback of this method is that unidirectional toolpaths are very inefficient and by having a constant spacing between each pass across the surface results in uneven scallop heights.

Zhu developed an algorithm to generate a rough cutting model of a given part[34]. The original part surface is transformed into a polygonal mesh and then offset by a desired amount or error. Any intersecting polygons are identified and corrected. This new offset model can then be used as a basis for toolpath generation to perform roughing on the part.

Lazoglu introduces an algorithm for toolpath generation with cutting force as the optimisation goal[35]. The free-form surface is divided into a uniform grid and the cutting forces between each point and it's eight neighbouring points is calculated. Then by analysing the minimum cost connections between all of the points and then connecting all of the individual connections so that no point is visited twice and every point is visited, an optimal toolpath with respect to cutting force can be generated.

Lee proposes a mesh based toolpath generation algorithm. The algorithm first generates an offset mesh from a free-form surface[36]. Any cutter location on this offset mesh will be interference free. The algorithm then produces a drive plane and creates cutter location points at each intersection between the offset mesh and drive plane. A new drive plane is then generated in such a way that the scallop height between the two drive planes is constant. This algorithm produced toolpaths with a lower error than toolpaths generated with an even spacing.

The problem with these methods is that they are not very flexible as they are usually designed to generate toolpaths for specific types of features or using a specific machining strategy. A universal algorithm is required which can generate a machining toolpath for any feature as defined by the ISO 10303-21 standard[37] and for any machining strategy as defined by the ISO 14649-11 standard[38].

2.2.2 Toolpath Generation for Five-Axis Machining

Five-axis toolpath generation has more variables to consider when compared to three-axis toolpath generation which results in a more complex solution space. Due to this complex problem, there has been a lot of research in this area. A state-of-the-art review was performed by Lasemi on CNC machining of free-form surfaces[39]. Three areas of free-form surface machining were explored which were toolpath generation, tool orientation identification and tool geometry selection. Also a comprehensive survey on toolpath interpolators, tool posture, gouging avoidance and adaptable geometric patterns for 5-axis milling was performed by Makhanov[40].

A 5-axis post-processor was developed by Takeuchi which took cutting locations generated by CAM software for three-axis milling and optimised the orientation of the tool at each location so that it was at a normal angle to the surface of the part[41]. The post processor would also take into account any overcuts between two points and insert additional points to overcome this issue.

Choi developed a search method which finds the optimal cutter orientations along a given 5-axis toolpath[42]. The yaw and tilt angles of the tool are used as the decision variables to determine which orientation of the tool is optimal with respect to the previous cutter location. The search method also takes gouging and collision errors into account. This is a brute force search method which generates optimal tool orientations but is very slow at finding each optimal value. This is because the search space is very large and there are multiple variables to optimise.

Algorithms were developed by Li to generate gouge-free, non-isoparametric 5-axis toolpaths

for free-form surfaces[43]. The cutter contact points are generated by representing the surface of the part as an approximation of polygons and finding the intersections between a vertical cutting plane and the polygons. The angles of the tool at each location is then set to the normal of the surface and adjusted if a gouging error would occur at the point.

Pi developed a "grind free" toolpath generation algorithm to reduce the size of the scallops created by the tool between each pass[44]. The method analysed the forward step between points and step over between passes to consistent and minimal scallop height. By combining the iso-parametric method of generating tool paths with the grind free method a much better surface finish was observed on the finished part.

Morishige uses configuration space to optimise the tool orientation of five-axis machining toolpaths[45]. By mapping the possible tool orientations into C-space and identifying the regions in which illegal tool orientations exist, a much more efficient method of searching for optimal tool orientations can be performed. Once the optimal tool orientations have been generated, intermediate cutting points are generated between each pair of consecutive points to reduce the angular change between points and increase the smoothness of the toolpath.

Bohez examines the relationship between the ideal linear interpolation between cutter location points in a toolpath to the actual travelled path between cutter location points on a five-axis machine[46]. By having a kinematic model of the machine, the generated grid to produce a toolpath from a surface can be adapted to account for the error previously mentioned.

Jun developed a configuration-space search method which optimises 5-axis machining toolpaths[47]. The aim of the method was to generate new toolpaths which did not contain collisions or gouging errors while optimising the smoothness of the toolpath. The method first identifies all of the legal tool orientations at each cutting location of the toolpath and then iterates along the cutting locations finding the tool orientation with least deviation from the previous point. The method produced results which optimised CAM generated toolpaths and ensured no gouging or collision errors were present in the toolpath.

Makhanov developed a grid optimisation technique which aims to aid in the generation

of better quality milling toolpaths[48]. The method uses a minimisation function with the scallop height and milling errors as the goal variables and takes zig-zag and spiral machining strategies into account. This algorithm is then applied to a pre-existing mesh grid generated using a marching method to adjust it accordingly.

A study was performed by Toh on the effects of milling strategy and cutting angle on the surface quality, cutting force and tool life[49]. It was found that with respect to roughing, the milling strategy and cutting angle had a negligible effect on the surface quality. However there wasn't enough data to find a correlation between roughing strategies and angles on the cutting force and tool life. More focus was put on finishing strategies and cutting angle. To extend tool life it was found that an angle of 15 ° to the normal was optimal. The best surface roughness was achieved at angles less than 10 ° to the normal.

Chen proposed a novel method of optimising the detection of tool interference in five-axis machining of sculptured surfaces[50]. First the surface is separated into convex and concave regions as convex regions do not suffer from tool interference. This technique improves the efficiency of detection as only the required regions are used in the calculation.

Tournier optimised the iso-scallop toolpath generation method by using a machining surface[51]. By modelling the toolpath as a surface, a more accurate calculation of the scallop heights can be performed. This method is more computationally intensive but resulted in a greater consistency of scallop heights produced by the machine tool.

Lu uses the concept of subdivision surfaces to produce an optimal polygonal mesh of a 3D model[52]. By analysing the errors between the polygonal mesh and 3D model and adjusting the resolution of the mesh in areas in which the error exceeds the tolerance, an optimal mesh for a given tolerance can be generated. This reduces the number of points required to model a part which therefore increases the efficiency of any toolpath generation method.

Anotaipaiboon introduces a new modification to the space-filling curves method of generating a grid for a free-form surface[53]. First a grid is generated with the scallop height constraints being taken into consideration instead of the kinematics error. Then the grid is modified into a space-filling curve or zig-zag path. Finally the number of points along

the path is adjusted to account for the kinematics error. This new method produced better results for complex free-form surfaces than the original space-filling curve method.

Lavernhe proposes a method of optimising 5-axis high speed machining by introducing the concept of a guide surface[54]. With high speed machining there is a need for monitoring the axis acceleration and jerk profiles and ensuring that they are within the machine limits. Therefore by generating a guide surface which specifies the tool orientation at any point on a surface, the acceleration and jerk of each axis can be kept within a tolerance. This idea was tested on a single pass and produced a toolpath with a reduced maximum acceleration and jerk profile when compared to following the surface normal along the same path.

He proposes a grid generation technique in which the size of the grid spacing varies across a free-form surface[55]. The grid spacing adjusts depending on the curvature of the free-form surface. In areas of greater curvature, the grid spacing reduces which increases the resolution of points in that area. This method assists in producing better quality toolpaths as more cutter locations are required in areas of greater surface change. The adaptive grid generation method reduced the magnitude of surface error when compared to the iso-parametric grid generation method.

Ren developed a method of generating spiral toolpaths for 5-axis milling by mapping a spiral toolpath onto a free-form surface[56]. The algorithm produced spiral toolpaths for a test part and produced a machined part with an acceptable surface finish.

Haranud proposes a toolpath generation method for 5-axis milling whereby the free-form surface is divided into subsections by clustering points which are near one another and have similar surface normals[57]. Then by determining the optimal toolpath for each subsection a better global toolpath can be generated. This method increases the surface quality of each local area of the free-form surface as having a global toolpath direction is not always optimal.

A grid generation method was developed by Anotaipaiboon which optimises the number and location of points to produce a toolpath with a minimal kinematic error[58]. The kinematic error is defined by the author as the deviation between the interpolating function from the actual trajectory. Therefore by manipulating the number and location of grid

points needed to generate a toolpath, a reduction in trajectory error can be made. The method starts by generating space-filling curves are for the free form surface. These are generated in such a way such that the maximum scallop height does not exceed the tolerance. Then a one dimensional grid is generated for each curve and the kinematic error is calculated for each grid by comparing the desired tool interpolation to an estimated trajectory created by a kinematic model of the machine. Grid points are then added to the one dimensional grid to reduce the kinematic error until it is below the tolerance. The results of the study seem promising. There was up to a 89% decrease in the kinematics error.

Can developed an iso-scallop toolpath generation algorithm for free-form surfaces defined by the STEP data structure[59]. The algorithm analyses the B-spline surface provided by the STEP data file and calculates the iso-scallop cutter contact points for that surface. The number of points are then optimised by calculating the error between cutter contact points and adjusting the distance between cutter contact points along a path. A tool selection algorithm was also proposed. The algorithm scans the entire surface of the part and identifies the region with the smallest radius. This is the maximum allowable radius of a tool to be used in machining the given surface.

Lasemi developed a method of improving the quality of a finished 5-axis part by analysing the surface of the manufactured part and adjusted the original cutting locations[60]. The surface of the manufactured part would be scanned using a CMM or laser scanner and then compared to the intended surface of the model. Error regions were then located and fed back into the original toolpath generation algorithm to account for the errors. A case study was performed which showed a definite reduction in surface error when compared to the original manufactured part.

The state-of-the-art review by Lasemi revealed a substantial decrease in the overall machining time and cost of machining due to the various toolpath generation techniques[39]. However, Lasemi states that many challenges still exist in generating quality toolpaths efficiently. The two main challenges still remaining are reducing the computation time required to generate toolpaths as well as sufficiently reducing the machining time required to machine

parts with free-form surfaces.

The survey by Makhanov revealed the various criteria that are used when generating toolpaths for 5-axis milling[40]. These criteria were: machining time, toolpath length, kinematics error, scallops, undercuts/overcuts, and rear gouging. The various constraints for generating toolpaths were also identified which were: machine axis limits, global gouging, acceleration and jerk of the tool. Many attempts have been made to optimise the various criteria mentioned when generating machining toolpaths but the survey discovered that no single solution which optimises all of the criteria mentioned above exists. However, the survey did identify the most promising methods of optimising each individual criterion which can be used to develop an overall solution.

2.3 Critique of the Literature

After reviewing the literature it is evident that there is a lot of interest and effort being made in the field of toolpath generation. A fully functioning method of generating optimised toolpaths for five-axis CNC milling would be extremely advantageous for the manufacturing industry to increase productivity. Many solutions have been proposed that aim to optimise certain aspects of toolpath generation such as cutting forces, surface quality, tool wear, tool engagement, scallop height and tool rotation.

There is a gap in the literature for a solution that optimises the toolpath length for five-axis milling. This solution would generate optimal toolpaths with respect to length while adhering to the constraints of avoiding tool collision and gouging and having the tool angle be as close to the surface normal as possible.

It can also be said that for three-axis milling, there are many distinct methods of generating toolpaths which depend on the geometry of the part or the machining strategy that is required. There is also no current method that considers more than one toolpath characteristic at a time in the optimisation of the toolpath. Therefore there is a gap in current research to find one toolpath generation method that would be independent of geometry and produce toolpaths with different machining strategies that would be optimised for a number

of toolpath characteristics.

There is also a lack of research in the generation of machining toolpaths with the specific aim of shortening the length of the toolpath. As commented by Anotaipaiboon, toolpath generation and optimisation is nothing more than finding the shortest path possible for a tool which can machine a part as required [61]. If the problem space was discrete rather than continuous, this would be equivalent to solving the travelling salesman problem with the tool tip as the salesperson and the cities representing the material that needs to be removed.

Chapter 3

Theoretical Framework

3.1 Introduction

This chapter will describe the theories and methods that were used to achieve the research goals and objectives defined in Chapter 1. This chapter will be split into a number of major milestones which can each be analysed in more detail.

3.2 Milestones

This section and the ones that follow will cover the major milestones that were defined for this research. The following list is an overview of the milestones that will be covered in detail in the following sections:

- Modelling Toolpath Generation as a Travelling Salesman Problem
  - STEP-NC Interpreter
  - Object Oriented Data Storage
  - Boundary Adjustment
  - Grid Point Generation

- Creating a Computational Platform for Solving the Travelling Salesman Problem
  - Developing the Genetic Algorithm
  - Optimising the Genetic Algorithm
- Validation of developed models

3.3 Modelling Toolpath Generation as a Travelling Salesman Problem

The first main objective is to develop a method of representing a part in such a way that it can be efficient for the toolpath generating algorithm to extract the required information. This modelling method will transform the part into a travelling salesman problem to be later solved. It is assumed that the part is defined by the ISO14649 standard[38], therefore an ISO14649 interpreter is required to extract all of the part information from the STEP-NC part 21 file[6].

3.3.1 ISO14649 Interpreter

The ISO14649 interpreter will parse through all of the lines contained within a STEP-NC part 21 file and extract all of the necessary information from it. This information will be stored in certain variables to be used later when rebuilding the part in the new representation method.

Figure 3-1 is a section of the part 21 code for an example part described in the ISO14649-11 standard. It can be seen that some of the lines describe the features of the part. From the code in Figure 3-1, object #17 describes the position and characteristics of a round hole and object #18 describes the position and characteristics of a closed pocket. This information along with that of the workpiece as described by the object #4 is all that is required to create a representation of the part. Therefore a parser will need to be developed that can parse through the lines of STEP-NC code and recognise features and workpiece characteristics. This requires a database to be written that contains all of the possible STEP-NC features as

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION((ISO 14649-11 EXAMPLE 1','SIMPLE PRORGRAM WITH A POCKET, AND A ROUND_HOLE'),'1');
FILE_NAME('EXAMPLE1.STP','2002-02-02',('YONG TAK HYUN','JOCHEN WOLF'),('WZL, RWTH-AACHEN'),$,'ISO 14649',$);
FILE_SCHEMA(('MACHINING_SCHEMA','MILLING_SCHEMA'));
ENDSEC;

DATA;
#1= PROJECT('EXECUTE EXAMPLE1',#2,(#4),$,$,$);
#2= WORKPLAN('MAIN WORKPLAN',(#10,#11,#12,#13,#14),$,#8,$);
#4= WORKPIECE('SIMPLE WORKPIECE',#6,0.010,$,$,$,(#66,#67,#68,#69));
#6= MATERIAL('ST-50','STEEL',(#7));
#7= PROPERTY_PARAMETER('E=200000N/M2');
#8= SETUP('SETUP1',#71,#62,(#9));
#9= WORKPIECE_SETUP(#4,#74,$,$,());
#10= MACHINING_WORKINGSTEP('WS FINISH PLANAR FACE1',#62,#16,#19,$);
#11= MACHINING_WORKINGSTEP('WS DRILL HOLE1',#62,#17,#20,$);
#12= MACHINING_WORKINGSTEP('WS REAM HOLE1',#62,#17,#21,$);
#13= MACHINING_WORKINGSTEP('WS ROUGH POCKET1',#62,#18,#22,$);
#14= MACHINING_WORKINGSTEP('WS FINISH POCKET1',#62,#18,#23,$);
#16= PLANAR_FACE('PLANAR FACE1',#4,(#19),#77,#63,#24,#25,$,());
#17= ROUND_HOLE('HOLE1 D=22MM',#4,(#20,#21),#81,#64,#58,$,#26);
#18= CLOSED_POCKET('POCKET1',#4,(#22,#23),#84,#65,(),$,#27,#35,#37,#28);
#19= PLANE_FINISH_MILLING($,$,'FINISH PLANAR FACE1',10.000,$,#39,#40,#41,$,#60,#61,#42,2.500,$);
#20= DRILLING($,$,'DRILL HOLE1',10.000,$,#44,#45,#41,$,$,$,$,$,#46);
#21= REAMING($,$,'REAM HOLE1',10.000,$,#47,#48,#41,$,$,$,$,$,#49,.T.,$,$);
#22= BOTTOM_AND_SIDE_ROUGH_MILLING($,$,'ROUGH POCKET1',15.000,$,#39,#50,#41,$,$,$,#51,2.500,5.000,1.000,0.500);
#23= BOTTOM_AND_SIDE_FINISH_MILLING($,$,'FINISH POCKET1',15.000,$,#39,#52,#41,$,$,$,#53,2.000,10.000,$,$);
```

Figure 3-1: An example part 21 code from ISO14649-11 Annex F.

described by the ISO14649 standard. The parser will then recognise the feature and extract all of the data within the line of part 21 code that describes the feature.

3.3.2 Data Storage

The data extracted from parsing a part 21 file can be stored in a object oriented program structure such as Java. An object can be created for every feature type described in the part 21 file as well as an object for the raw workpiece which is also described within the part 21 file. The object can contain all of the information required to define the features such as position and dimensions.

3.3.3 Creating the Model

A model of the part can be created from the information stored in the objects defined in the previous section. There are many types of models that can be used to represent the part. A constructive solid geometry(CSG) technique can be adopted to represent the part. The initial construction would be the surface of the raw workpiece which would then have the

subsequent features removed from it.

Another technique to represent the part could be using boundary representation(B-Rep). This is a collection of faces, edges and vertices that create a surface of the part being modelled. The model would start with the raw workpiece and then the features would be added to the model.

Both the CSG and B-Rep techniques represent the part as a set of surfaces between edges or vertices. The nature of the travelling salesman problem is that of finding an optimal path between a number of points. Therefore if all of the points are along the boundaries or edges of the part then a machining toolpath cannot be generated using these two techniques.

A more appropriate method would be to have the part represented in a grid format where each cell in the grid would represent a point on the part. This could be done using a 3D grid with a resolution sufficiently small to not lose any important feature details. Each block in the grid would behave as a 3D pixel (also known as a voxel). The travelling salesman problem can now be performed on all of the voxels that need to be removed from the raw workpiece to produce the finished product as described by the part 21 file.

Although it is now possible to generate a viable toolpath from the points in the 3D grid, solving a 3D travelling salesman problem is far more difficult than solving a 2D problem. Therefore the 3D grid can be replaced with many layers of 2D grids which are spaced apart according to the cut depth of the tool. Converting the 3D travelling salesman problem into the 2D layered problem will also more accurately resemble a machining toolpath. This is due to the cutting tool only being able to remove material at a certain depth with each pass of the material. Figure 3-2 is an example part with a simple feature.

By analysing the part 21 file for the example part in Figure 3-2, the dimensions and position of the feature on the part can be extracted. A 2D grid can be created with dimensions slightly larger than the part itself to allow for tool movement around the part. The resolution of the grid will depend on the complexity of the features (i.e. does it contain curved features). For the example part the resolution will be relatively low.

An example of how the grid structure would look can be seen in Figure 3-3. The red cells

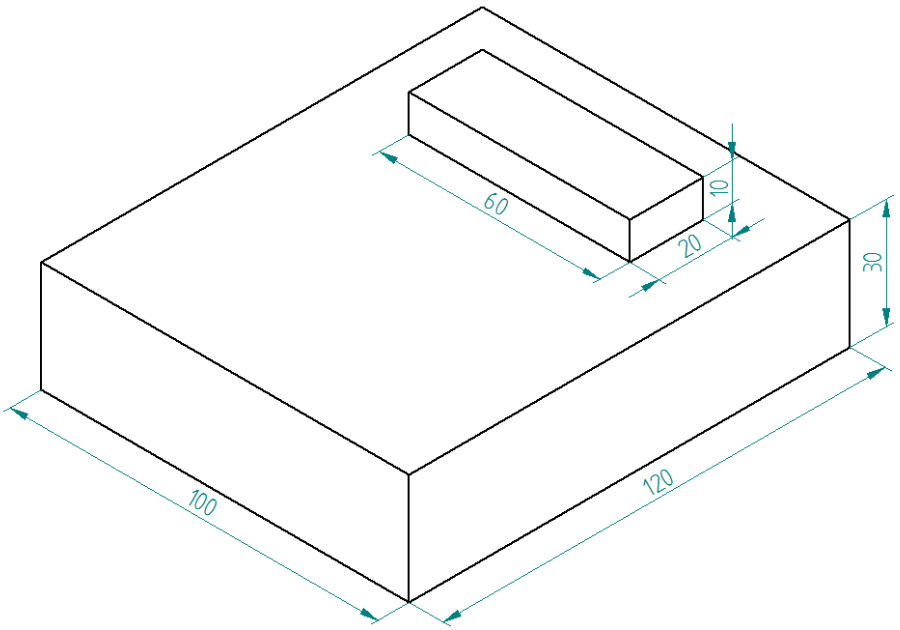


Figure 3-2: A simple example part.

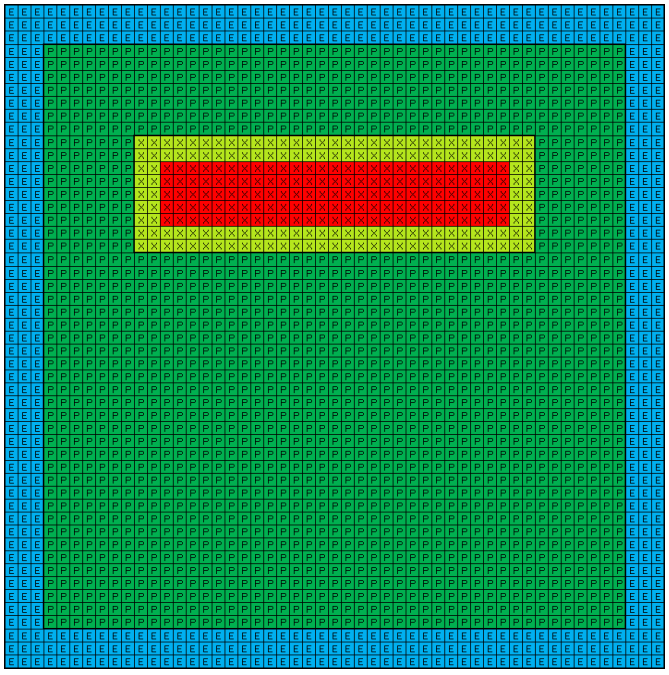


Figure 3-3: An example of a 2D Grid for the part in Figure3-2.

in the grid would represent points that shouldn't be used in the toolpath as this is the feature that needs to be machined around. The light green cells represent a region around the feature with a distance equal to the tool radius. This ensures that The dark green cells represent all of the points on the surface of the raw workpiece that need to be removed. Therefore all of these points would be included in the machining toolpath. The blue cells represent the space around the part which can be used if a certain machining strategy requires these points to be visited.

3.3.4 Data Point Reduction/Optimisation

The second objective is to have an algorithm that can reduce the number of data points required to represent a part. Large numbers of data points does not always equate to a better representation of the part. Having an even distribution of points over all of the parts surfaces would be an example of redundant data points. Only data points along the boundaries of features and the part are required to represent a part, however, more points are required to generate a valid toolpath to machine the features of the part. Therefore the number of points required to generate viable toolpaths for a part will be somewhere in between these two scenarios and will depend on the radius of the tool, tool overlap and the machining strategy.

It is useful to reduce the number of data points as much as possible for a more efficient performance of the heuristic algorithm used to generate tool paths from these points. Decreasing the number of data points decreases the computational effort required to generate an optimal path through the points. This can easily be seen as the number of possible paths through N number of points is N!.

$$Number of Solutions = N! \tag{3.1}$$

Therefore decreasing the number of points dramatically reduces the search space that the heuristic algorithm has to work with and decreases the time and effort required to find an optimal solution.

The following sequence of figures illustrates the various stages in the process of reducing

the number of data points required to represent a part.

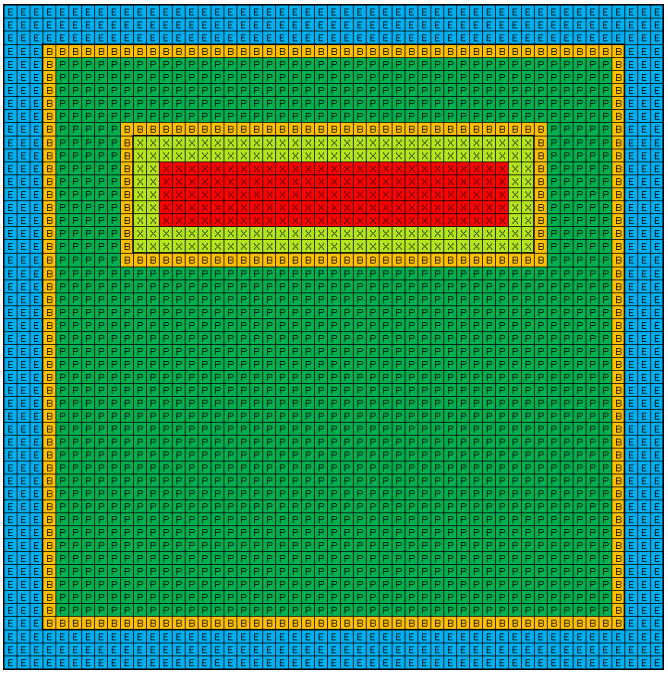


Figure 3-4: Visualising the grid system for an example part with boundary points.

Figure 3-4 illustrates the grid point system for the example part in Figure 3-2. This grid differs from the one in Figure 3-3 in that it has orange grid cells which represent points which are located on the boundary of the part or the boundary of a feature.

The first phase of the data point reduction algorithm would remove data points in such a way that no two data points would neighbour each other on the grid with the exception of the data points on the boundary. This reduction can be seen in Figure 3-5. This process by itself already reduces the number of data points by approximately half of the original amount. The data points on the boundaries of the features as well as the part itself have been kept as it is important to keep a greater number of data points in close proximity to areas on the part with a change in surface.

The second phase of the data point reduction algorithm would remove more data points ensuring that no data points are within a tool's radius distance from another data point. This can be seen in Figure 3-6. Even though the data points have been reduced significantly, any generated toolpath would still machine the desired part as long as all of the data points

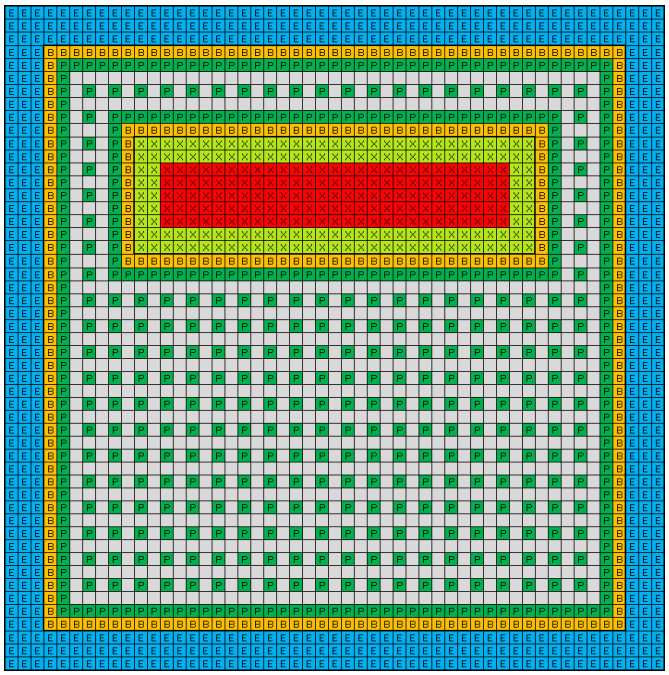


Figure 3-5: First step of the point reduction process.

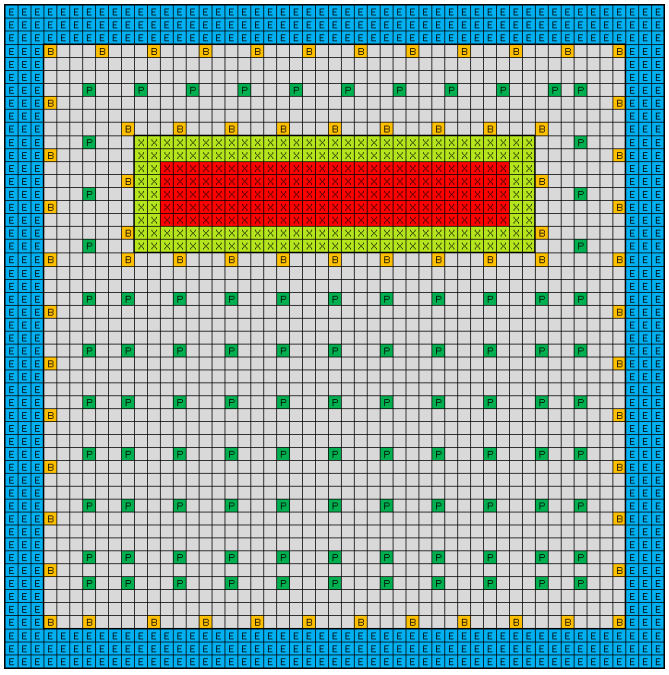


Figure 3-6: Second step of the point reduction process.

are visited within that generated toolpath.

At this stage any path through the points which does not cross a boundary will be a valid toolpath which will machine the given feature fully. The optimisation of a path through a set of points can be seen as a travelling salesman problem.

3.4 Creating a Computational Platform for Solving the Travelling Salesman Problem

The second main objective is the generation and optimisation of machining toolpaths whilst adhering to various machining strategies. This will be done by taking the travelling salesman problem produced in the primary objective and solving it so that it fits the set of requirements needed for a valid machining toolpath. The travelling salesman problem is a problem with a very large solution space and is very difficult to solve analytically. Therefore a computational platform will be developed which will contain a meta-heuristic to help solve this optimisation problem.

3.5 Toolpath Planning Modelled as a TSP

The traveling salesman problem is a very old problem which can be defined as finding the shortest path between a set of N points[62]. Each point can only be visited once and all the points in the set need to be visited. Mathematically the problem can be described as follows[63]. Given a cost matrix $D = (d_{i,j})$, where $d_{i,j}$ = the cost of going from city i to city j, (i, j = 1, 2, $\cdots$, n), find a permutation $P = (i_1,\ i_2,\ i_3,\ \cdots\ ,\ i_{\mathrm{n}})$ of the integers from 1 through n that minimises the quantity

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} x_{i,j} \tag{3.2}$$

which is subject to the following conditions

(a) $x_{\mathrm{i,i}} = 0$

(b) $x_{\mathrm{i,j}} = 0, 1$

(c) $\sum_i x_{\mathrm{i,j}} = \sum_j x_{\mathrm{i,j}} = 1$

(d) and for any subset $S = i_1, i_2, \cdots, i_r$ of the integers from 1 through n,

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_{\mathrm{r\text{-}1}} i_{\mathrm{r}}} + x_{i_{\mathrm{r}} i_1} \begin{cases} < r \text{ for } r < n. \\ \leq n \text{ for } r = n. \end{cases} \tag{3.3}$$

Condition (a) ensures that a point cannot connect to itself. Condition (b) ensures that each point is visited only once, condition (c) ensures that all the points are visited and condition (d) ensures that no sub-tours being generated. There are two types of traveling salesman problem, if $d_{i,j} = d_{j,i}$ then the TSP is symmetrical and if $d_{i,j} \neq d_{j,i}$ then the TSP is asymmetrical[64]. The asymmetrical TSP is more challenging to solve as the asymmetric TSP has to be converted into a symmetric TSP which doubles the size of the cost matrix[65]. The toolpath generation problem could be viewed as an asymmetrical problem if the goal of the toolpath considers conventional or climb milling strategies. This is due to some directions of travel would be preferred to others. However, in this research conventional and climb milling will not be considered in generating toolpaths and therefore the problem will be viewed as a symmetrical TSP.

TSPs has been used to describe many path planning problems including job shop planning[66], robot arm position control[67], pick-up and delivery of products[68] as well as many others. Toolpath generation can be seen as a TSP much in the same way as any other path optimisation problem. The problem of generating and optimising a toolpath can be characterised as a TSP by representing the feature as a set of points that the tool has to traverse in order to machine the part. The traveling salesman problem can even be performed on a freeform surface by calculating the distance between two points across a surface using the algorithm developed by Maekawa[69]. The cost of traveling between two points would not be the linear distance but the parametric distance between the two points. Any path that traverses through all the points in the set can be seen as a toolpath. Any subsequent path that is generated and produces a lower value for Equation 3.2 can be seen as an optimisation of that

path.

Although the type of TSP that will be developed when modelling the toolpath generation and optimisation problem will be a variation of the traditional TSP, this review will cover the various methods used in solving the traditional traveling salesman problem by using analytical, heuristic and meta-heuristic techniques. The method of solving this new variation of TSP can be adapted from the current methods of solving the traditional TSP.

3.6 Solving the Traveling Salesman Problem

A survey of methods used to solve the traveling salesman problem was performed in 1968 by Bellmore and Nemhauser[70]. This survey covered the early algorithms and heuristics developed and compared the performance between these methods. An updated survey was then performed by Johnson and McGeoch in 1997[71]. As computing power had significantly improved since the first survey, the complexity of the methods used in this newer survey increased which resulted in much better performance in solving the TSP. The updated survey included the best performing heuristics as well as meta-heuristics. Another survey was performed by Chauhan et al. in 2012[72]. The most recent survey was performed by Jiang et al. in 2014[73]. This section will cover some of the algorithms discussed in these surveys as well as continued development in TSP solving algorithms since these surveys were published.

The simplest yet least efficient method of solving a TSP is by calculating all of the possible path lengths and then choosing the path with the minimum length[74]. This method is known as the brute force method. It is only feasible for very small problems and can take an extraordinary amount of time for larger problems[75]. A more efficient method of solving the TSP analytically is by using the branch-and-bound method[76, 77, 78, 79]. This method uses a tree to represent the various path permutations. The lower bound and upper bound values of each branch is then calculated and compared to analyse which branches can be discarded. This process is repeated until the branch with the lowest value for the lower bound remains which will contain the shortest possible path. Some variations of the branch-and-bound method have been developed to improve the efficiency of finding the optimal

solution[80, 81, 82].

The challenge with the traveling salesman problem is that the number of solutions to each problem is a factorial of the number of points to be visited. Hence finding an analytical solution to the traveling salesman problem is only feasible for problems with only very few points. Already at 15 points, the number of solutions is over 1.3×10^{12}. Therefore heuristics are used to solve the problem in a more time efficient manner.

3.6.1 Heuristics

A heuristic is a problem solving process which learns and adapts from previous experience and knowledge to obtain a solution[83]. They are a lot quicker than standard analytical problem solving techniques when the problem space is very large, however the disadvantage to using heuristics is the quality of the solutions generated. The solutions generated will most likely be approximate or near optimal depending on the heuristic used and the structure of the heuristic[84].

The nearest neighbour algorithm is a simple and efficient heuristic for solving the TSP[85, 70]. The heuristic works by starting at a random point in the set and then determines the next closest point to it and adds that point to the path. This process continues until all of the points have been visited. This heuristic can be improved by starting the algorithm from each point in the set to find the best possible path using the nearest neighbour algorithm. A similar heuristic to the nearest neighbour algorithm is the greedy algorithm[86]. This algorithm starts with the edge with the lowest cost in the set and then removes all the edges connected to the two chosen points. This process is then iterated until all of the points have been inserted into the path.

The Clarke-Wright heuristic works in a different way to the previous two[87]. A point in the set is chosen to be the "hub" point. The path then follows each other point in the set but visiting the hub between each pair of points. The algorithm then ranks the savings that could be obtained if the path between any two points were to skip the hub. The hub is then removed from each sub path in order of most cost savings per iteration until only two

points are connected to the hub and a full path is created. This algorithm produces better results than the Greedy and Nearest Neighbours heuristics but requires more computational effort[86].

The Christofides algorithm is another heuristic developed to solve the TSP[88]. This algorithm starts by creating a minimum spanning tree of the set of points. On all of the nodes with odd degrees, perform a minimum cost matching process to convert these odd nodes into even nodes. Finally convert the path into a Hamiltonian path by skipping visited nodes.

These four heuristics produce solutions to the TSP within 10-15% of the optimal path[71]. These heuristics are commonly referred to as approximation algorithms in the literature. Johnson performed an experimental analysis of these four approximation algorithms and found that although the Christofides algorithm performs the best at around 10% of the optimum, it is over five times slower than the Greedy algorithm which produces results at around 16% of the optimum[89].

Due to the lack of quality in the solutions they generate on their own, they are usually used to construct an initial path as a baseline for other heuristics or meta-heuristics to further improve upon the results from these four heuristics. It is also a lot quicker to produce results within 10-15% of the optimal solution with these heuristics than to start with random paths and optimise them to the same level using better heuristics or meta-heuristics.

The following set of heuristics are known as local search algorithms. These heuristics take a given path for a TSP and attempt to modify them iteratively by exchanging edges until no further improvement can be found. These heuristics are also known as neighbourhood search algorithms as one path can be described as a neighbour to another path if it only requires one move to switch between the two.

The simplest local search algorithm is the 2-opt exchange algorithm proposed by Croes[90]. The algorithm works by removing two edges from the path and rejoining the path by using the other two possible edges that will create a full path without subpaths. The algorithm scans all the possible exchange moves and calculate the cost that could be saved with each

one and performs the best exchange for each iteration. This process continues until no further improvements to the path are possible. Figure 3-7 shows an example of a 2-opt exchange.

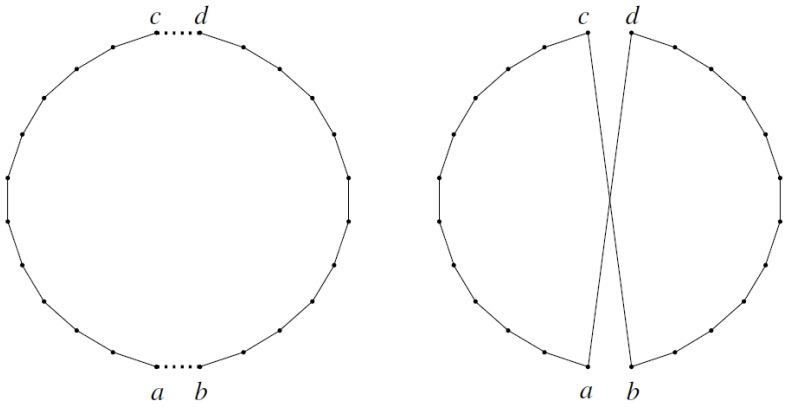


Figure 3-7: An example of a 2-opt exchange with the resulting path.

The 3-opt exchange algorithm was proposed by Bock and developed by Lin[91, 63]. This algorithm works in the same was as the 2-opt exchange algorithm but removes three edges as opposed to two. With the 3-opt method there are two possible ways to exchange the edges so that no subpath is created. These two possible exchanges can be seen in Figure 3-8. This method produces more optimal results compared to the 2-opt method but due to the increased number of possible exchanges this method requires more computational effort.

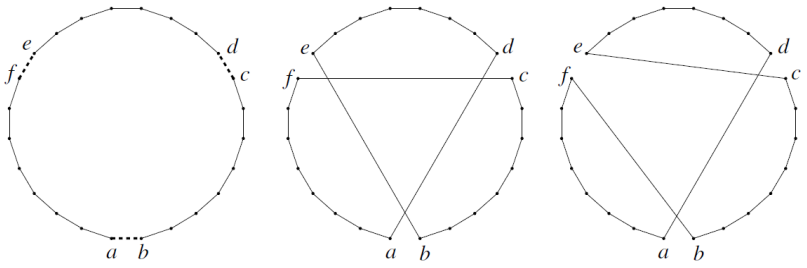


Figure 3-8: An example of a 3-opt exchange with the two possible exchanges.

Bentley also describes a similar local exchange algorithm as being 2.5-opt[92]. This algorithm removes a point from the path and inserts it elsewhere in the path that would provide the best cost savings for the path. This process also removes three edges from the path and inserts three new edges into the path in a similar manner to the 3-opt exchange algorithm. An example 2.5-opt exchange can be seen in Figure 3-9 where point B is removed from the

D-B-E subpath and places into the A-B-C subpath. The 2.5-opt method requires only slightly more computing power than the 2-opt method but produces substantially better paths.

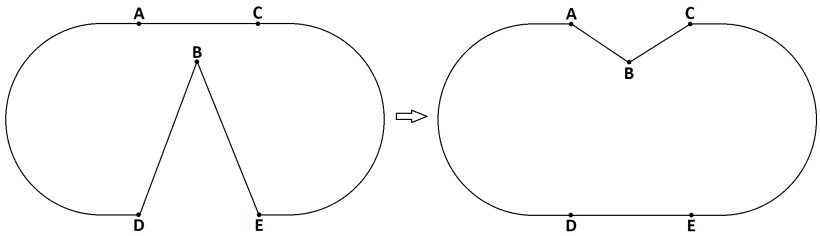


Figure 3-9: An example of a 2.5-opt exchange.

One of the best performing local search algorithms was developed by Lin and Kernighan in 1973 and is known as the Lin-Kernighan(LK) heuristic[93]. This is an adaptive algorithm that combines the 2-opt and 3-opt exchange methods and calculates which of the two exchanges produce the best possible cost savings for the path. It was also proposed to keep a list of both the added and deleted edges so that the heuristic does not attempt to add an edge that was deleted or delete an edge that was previously added. The LK heuristic was the best performing local search heuristic for solving the TSP up until 1989[71]. Johnson proposed a variation of the LK heuristic which only considered maintaining the list of added edges[94] as opposed to only maintaining a list of deleted edges as proposed by Papadimitriou[95]. It was found that the method of only maintaining the list of added edges performed much better than maintaining both lists or just the deleted edges list. This was due to the increase in search space of the algorithm and the fact that added edges were more important than deleted edges as some deleted edges could still appear in the final iteration of the algorithm[71].

The most notable adaptation of the LK heuristic was developed by Helsgaun in 2000 known as the Lin-Kernighan-Helsgaun(LKH) heuristic[96]. This method uses one-tree approximations of the set of points to create a list of the best edges for each point. This is combined with the possibility for k-opt moves where $2 \leq k \geq 5$. This widens the search space of the algorithm allowing for better results at the expense of taking longer. Using the one tree approximation reduces the computational effort required compared to regular exchange algorithms. Helsgaun further developed the LKH heuristic in 2009 to allow for k-opt moves

where $2 \leq k < N$ as well as a partitioning and merging algorithm to break the overall set of points into subproblems which can be solved individually and rejoined. This new adaptation is known as the LKH-2 heuristic and when combined with an approximation algorithm to generate start tours it has been shown to outperform all other developed heuristics to date[73].

There has been a lot of development in the field of heuristics with respect to solving the traveling salesman problem. Preiss states that it is important for a heuristic to have the capability of backtracking so as not to converge on a non-optimal solution[3]. However, the most efficient method used to solve the traveling salesman problem are evolutionary algorithms which avoid backtracking. They overcome this issue by being able to explore a large solution space very efficiently and converge on an optimal solution without then need to backtrack. The main types of evolutionary algorithms used to solve the traveling salesman problem are genetic algorithms, ant colony optimisation systems and simulated annealing which will be covered in the next section of the literature review.

3.6.2 Meta-heuristics

Meta-heuristics are a higher level heuristic which generate approximate solutions to optimisation problems without having to consider every possible solution. Meta-heuristics are generally used for problems that have a very large solution space that cannot be efficiently explored using heuristics or analytical methods[97]. Many types of meta-heuristics have been used in solving the TSP due to its very large solution space. Ant colony optimisation, genetic algorithms, tabu search, simulated annealing and neural networks have all been used to solve the TSP[98]. The three best performing meta-heuristics were genetic algorithms, ant colony optimisation and simulated annealing[71].

(i) Genetic Algorithms

Genetic algorithms are used as meta heuristic problem solving technique in many types of industries and has many applications[99]. It is modelled after the process of biological

evolution of species by natural selection. The algorithm is based off of the work done by Fraser in 1957 where the evolutionary process was simulated using computational methods[100, 101]. This was further developed and adapted into an algorithm to solve mathematical problems by Holland in 1975[102]. Genetic algorithms are an efficient method of converging on an optimal solution when there is a large problem space to analyse.

In this process, individuals improve their ability to survive in an environment over many generations by imparting beneficial characteristics which allow a greater chance of survival to their offspring. These characteristics are defined by the genetic information contained within an individual.

New genetic material can be introduced into a system by the means of genetic mutation. This allows for new characteristics to be observed and improvements to be made to previous characteristics.

Genetic algorithms to tackle the traveling salesman problem were first developed by Brady in 1985[103]. By modelling the system as the environment and by defining the various solutions to the traveling salesman problem as individuals with each gene representing a point in the path, the optimal path could be evolved from an initial population of random non-optimal solutions.

Genetic algorithms have a standard structure:

- Step 1: Create initial random population
- Step 2: Calculate fitness of individuals
- Step 3: Selection of individuals for reproduction
- Step 4: Crossover of genes
- Step 5: Mutation
- Repeat steps 2-5

Each of these steps can be observed in nature and need to be modelled in a genetic algorithm for it to function properly. Many operators have been proposed for the various

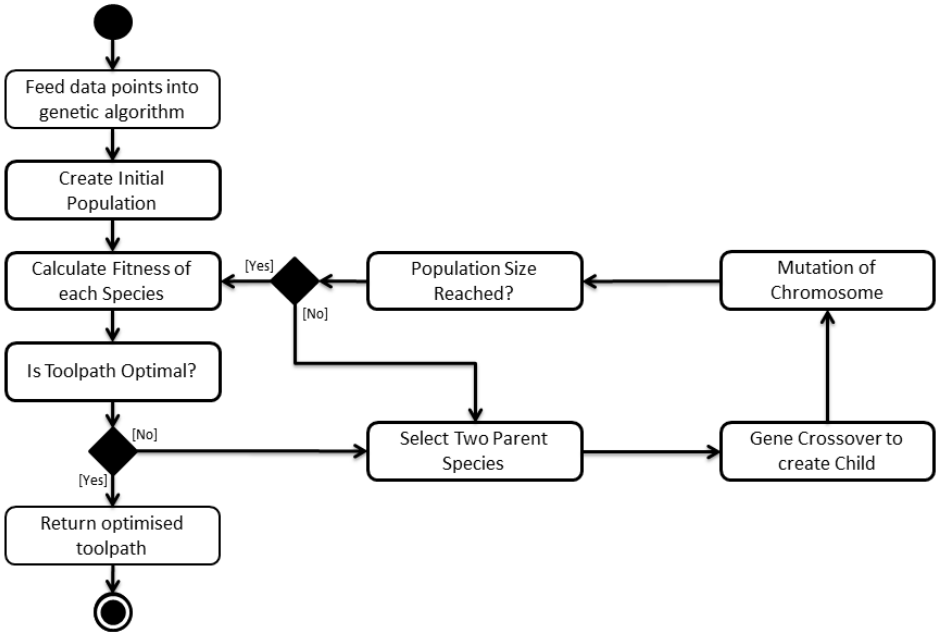


Figure 3-10: A flowchart showing the structure of a typical genetic algorithm

components of a genetic algorithm. Some operators work better than other in all problems were as some only outperform other in specific problems. As the order of the genes are important when solving the TSP with genetic algorithms, the operators that will be reviewed

(a) Species Selection To create a new generation from an existing population, a pair of individuals need to be chosen to reproduce. One or more individuals can be used to do this in different ways. A survey and experimental analysis of selection operators for genetic algorithms was performed by Blickle and Thiele in 1995[104]. This section covers the operators analysed in this survey along with other operators developed after the survey was performed.

Miller demonstrated how the performance of the genetic algorithm is dependent on the selection pressure applied to the population by the selection operator[105]. If the selection pressure is high, the genetic algorithm will favour species with higher fitness levels and converge much faster than if the selection pressure is low. The drawback is that with high selection pressure, the probability of early convergence increases due to the narrowing of the

search space. Therefore it is important to balance the selection pressure for the required problem.

The simplest method of achieving this is to select the fittest individual of each generation and sufficiently mutate the genetic information to simulate reproduction. Although this is the quickest method as it bypasses the crossover stage, it has a very high chance of converging on a non-optimal solution as this method narrows the search space very quickly[106].

The next best method is to have the top N fittest individuals of a population reproduce, this is called tournament selection[105]. A subset of size S is selected from the total population. This subset is then filled randomly with individuals from the population which then compete to be in the selection pool. The winner will be the individual with the highest fitness. The amount of selection pressure is dependent on the size of S. This method reduces the search space relatively quickly and performs better than the previous method.

Mühlenbein uses a truncation selection operator which sorts the population into order by their fitness and then selects a proportion of the population to be copied and mutated to return the population to its original size[107]. The truncation operator has only been used in a small number of genetic algorithms since its establishment. This is probably due to the high selection pressure it places on the population which can cause early convergence in the genetic algorithm[104].

Another method is to rank the population according to the individual's fitness value and then use a function that randomly selects the parents which is weighted towards the rank. This method is called the Linear Ranking Selection Operator[108, 109]. Therefore the higher the rank, the more likely an individual will be chosen. This method is an improvement to the previous method as it still favours better performing individuals to reproduce while accepting lower performing individuals to reproduce to avoid premature convergence on a local maxima.

A similar method is to use a function that selects parents according to their relative fitness. This method is known as roulette wheel selection or fitness proportional selection[102]. This method ensures the least reduction of the search space while still favouring better performing individuals. Sengoku uses this method in a TSP solving genetic algorithm with

an added function where similar individuals are eliminated[110]. This removes any duplicate individuals which can cause premature convergence. If the top two performing individuals in a population have the same chromosome then crossing over these two individuals will not create a new chromosome. Therefore by eliminating duplicate individuals, gene crossovers will be performed on differing individuals and will create new chromosomes which keeps the solution space wider for longer.

A new operator was developed by Kureichick which helps a genetic algorithm avoid getting stuck in local minima[111]. If many of the individuals in a population have the same length, it can be assumed that a local minima has been reached. If this is the case, then all of the individuals with the same length are modified except for one individual. This prevents early convergence and helps the genetic algorithm to generate an optimal solution.

The experimental analysis of selection operators performed by Blickle and Thiele in 1995 showed that the Tournament selection and Linear Ranking selection were the best performing selection operators for genetic algorithms[104]. This finding is reinforced by the amount of genetic algorithms in the literature that use these two operators and the lack of development of new selection operators. The paper also analysed the effects of selection pressure on population variance and algorithm convergence which can be used to decide which operator is more suitable for a problem being solved using a genetic algorithm.

(b) Gene Crossover In non-ordered chromosomes, gene crossover is a straightforward process of randomly selecting a gene from each parent or by taking an average value of the two. This cannot be done with ordered chromosomes as the individual's fitness is dependent on the order of the genes contained within a chromosome. Duplicate gene values are also generated using the non-ordered crossover methods, therefore it is best to use crossover techniques designed for ordered chromosomes[112]. Üçoluk discusses the various techniques that can be used for crossing over genes in an ordered chromosome genetic algorithm[113]. The first technique is disqualification, where invalid solutions created by ordinary crossover operators are given extremely low fitness values. The second technique is reparation, which feeds the invalid solutions into an intermediate process which repairs any duplicate or omitted

points. The third technique is to use only special operators which are designed for ordered chromosomes which produces only valid chromosomes. Disqualification is a very inefficient technique as a lot of computational effort is spent generating invalid solutions. Reparation does generate valid chromosomes however as it requires an intermediate stage it is very time consuming compared to using special operators as these generate valid chromosomes without the need of a secondary stage.

The crossover operator that is used for the genetic algorithm depends on the format of the chromosome in terms of the representation of the genes. There are a number of different types of gene representations each with crossover operators which function for that representation. In a review performed by Larrañaga[114], the various representations are analysed. The following is a brief summary of the various representations with the most common crossover operators for each.

(b.1) Binary Representation One way of representing the gene values is by converting the number of the city into a binary value. For example, if a chromosome contained the genes 0-1-2-3-4-5, then the equivalent in binary representation would be:

(000 001 010 011 100 101)

To cross over the genes in the chromosome, the classical crossover was proposed by Holland in 1975[102]. A crossover point is chosen and all of the genes beyond that point is copied from one parent and all of the genes before that point is copied from the other parent. This crossover technique creates paths with duplicate points or omitted points which do not represent a valid solution for the TSP. Also the classical mutation operator was not very successful as this often created invalid solutions for the TSP. The classical mutation operator would randomly select a bit in the chromosome and invert its value. For example, if the first bit was mutated in the previous example chromosome then the outcome of that mutation would be:

(100 001 010 011 100 101)

Which is an invalid solution as the 100 point appears twice and the 000 point does not appear at all in the chromosome. This method was applied to the TSP by Lidd in 1991 but only

yielded optimal results for low numbers of cities[115].

(b.2) Path Representation Another way to represent the cities in the chromosomes is to assign a numerical value to each city and the same numerical value to the gene corresponding to the position of that city in the chromosome. Therefore a path with the genes 0-1-2-3-4-5 will have the following chromosome:

(0 1 2 3 4 5)

This representation has many operators that have been developed for conserving the order of the genes as well as avoiding duplicate genes and omitted genes. The following is a list of the crossover operators found in literature:

- Order Crossover (OX1)[116]
- Order Based Crossover (OX2)[117]
- Variations of Ordered Crossover (VOX)[118]
- Partial Mapped Crossover (PMX)[119]
- Genetic Edge Recombination Crossover (ER)[120]
- Cycle Crossover (CX)[121]
- Position Based Crossover (POS)[117]
- Heuristic Crossover (HX)[122]
- Sorted Match Crossover (SMX)[103]
- Maximal Preservative Crossover (MPX)[107]
- Edge Assembly Crossover (EAX) [123]
- Voting Recombination Crossover (VR)[124]
- Alternating-Position Crossover (AP)[125]
- Relative Order Crossover (ROX) [112]

The crossover algorithms mostly function in one of two methods. The genes can either be crossed over and then analysed to repair any duplicate or omitted genes, or the new chromosomes are created in such a way as to avoid duplication or omission.

(b.2.1) Partially Mapped Crossover The partial mapped crossover operator was developed by 1985 by Goldberg and Lingle[119]. This operator takes two parent chromosomes and selects a random section of the chromosome and maps the values of the genes to the same section on the other parent. For example, if the two following parents are chosen:

Parent 1 - (0 3 7 5 4 1 8 2 6)

Parent 2 - (0 5 8 2 6 3 1 4 7)

If the 5-4-1 section of genes are chosen for mapping from Parent 1 and mapped onto the same section of genes in Parent 2:

Parent 1 - (0 3 7 $\boxed{5\ 4\ 1}$ 8 2 6)

Parent 2 - (0 5 8 $\boxed{2\ 6\ 3}$ 1 4 7)

Then it can be seen that gene 5 maps to gene 2, gene 4 maps to gene 6, and gene 1 maps to gene 3. Now the two different offspring can be created by rewriting the parent genes with the mapping of the genes as previously specified.

Offspring 1 - (0 **1** 7 $\boxed{2\ 6\ 3}$ 8 **5** **4**)

Offspring 2 - (0 **2** 8 $\boxed{5\ 4\ 1}$ **3** **6** 7)

The emboldened genes indicate which genes have been altered due to the mapping. This method ensures the order of the genes are kept as well as inheriting a section of the chromosome from one of the parents while preserving as much of the rest of the chromosome of the other parent.

(b.2.2) Edge Recombination Crossover A lot of information can be obtained by looking at the genetic edges of a chromosome. The edge of a gene looks at the spaces between certain genes. Consider the following two parent chromosomes:

Parent 1 - (0 3 7 5 4 1 8 2 6)

Parent 2 - (0 5 8 2 6 3 1 4 7)

If the gene value of 4 is chosen for this example, then the genes on either side of gene with a value of 4 is examined in both parents.

Parent 1 - (0 3 7 [5] **4** [1] 8 2 6)

Parent 2 - (0 5 8 2 6 3 [1] **4** [7])

In Parent 1, gene 4 has an edge with gene 5 and gene 1. In parent 2, gene 4 has an edge with gene 1 and gene 7. Therefore the genetic edges of gene 4 would be 1, 5 and 7. To perform the Edge Recombination Crossover, all of the edges of two parent chromosomes need to be analysed. All of the edges for the example parents can be seen in Table 3.1.

Gene Value	Genetic Edges
0	3, 5, 6, 7
1	3, 4, 8
2	6, 8
3	0, 1, 6, 7
4	1, 5, 7
5	0, 4, 7, 8
6	0 ,2, 3
7	0, 3, 4, 5
8	1, 2, 5

Table 3.1: Genetic edges for example parent chromosomes

Now that the genetic edges have been analysed, a new offspring can be created using the edge information. By starting with a random gene value, for example 2, the gene that follows will either have a value of 6 or 8 according to Table 3.1. Therefore the gene value is randomly chosen from the selection of edges available. This process continues until all of the gene values have been chosen. These are the two main types of crossover operators for path representation chromosomes.

(b.3) Adjacency Representation Another method of representing the cities in a tour of a TSP is the adjacency representation. In this representation each gene value represents a path between the city with a value of the position of the gene, and a city with a value of the value of the gene. For example the chromosome with city values:

(0 3 7 5 4 1 8 2 6)

Can be written in adjacency representation in the following way:

(3 8 6 7 1 4 0 5 2)

Which can be better understood by expressing each value in the chromosome as a sub-tour between two points:

(0→3, 1→8, 2→6, 3→7, 4→1, 5→4, 6→0, 7→5, 8→2)

By following each path, a full tour around all of the points can be made. There are a number of crossover operators which can be used with the adjacency representation:

- Alternating Edge Crossover (AER)[126]
- Sub-tour Chunks Crossover (SCX)[126]
- Heuristic Crossover (HC)[127]

The most common chromosome representation used in the literature is the path representation. There are many well established operators for this chromosome representation for both crossover and mutation. This is due to the fact that because of the simple nature of the chromosome structure (i.e. each gene value corresponds to a particular point) the crossover and mutation operators are a lot simpler and produce viable individuals far more frequently than operators used in the other chromosome representations[120].

A comparative study was performed by Starkweather compares 6 different crossover operators on a 30 city traveling salesman problem. Each operator was fine tuned with respect to all of the other variables in the genetic algorithm (i.e. population size, mutation rate etc.). It was found that the order crossover operator outperformed all of the other crossover operators. It was discussed that operators which most preserved the order of the chromosome performed better. As only one traveling salesman problem was tested it is not conclusive whether the ordered crossover operator is the most efficient.

Abdoun discusses and analyses a number of crossover operators for the traveling salesman problem[128]. These include: Uniform crossover, Cycle crossover, Partial mapped crossover,

Uniform partially mapped crossover, Non-wrapping ordered crossover, Ordered crossover, Crossover with reduced surrogate and Shuffle crossover. The various crossover operators were used to find a solution to the BERLIN52 traveling salesman problem. All of the other genetic algorithm operators and variables remained constant so as to compare the performance of the various crossover operators. It was found that the Ordered crossover and Non-wrapping ordered crossover had a very similar performance with both operators performing significantly better than all of the other tested operators. It cannot be said that these two operators are the most efficient operators for all TSP scenarios as only the BERLIN52 scenario was tested.

A study was carried out by Yoon to analyse the effects of combining various crossover operators and compare the performance of different combinations with respect to using only one crossover operator[129]. It was found that there was an increase in performance when combining various crossover operators but it was not conclusive as to which combination produced consistently better results when tested on various problems.

(c) Mutation Operator A vital part of the genetic algorithm is the mutation operator. Mutation occurs after a new individual is created using a gene crossover operator. This ensures that new genetic material is introduced into the population with each generation and prevents early convergence on a non-optimal solution. Early convergence is avoided with mutation as the search space is kept from diminishing when new gene combinations are inserted into the population[130]. There are many types of mutation operators that exist in literature. However, most of the operators are used in conjunction with the path representation of chromosomes as mutations in this particular representation produce the most viable and "legal" toolpaths[114]. The following is a list of common mutation operators found in literature:

- Displacement Mutation (DM)[131]
- Exchange Mutation (EM)[132]
- Insertion Mutation (ISM)[133]
- Simple Inversion Mutation (SIM)[102]

- Inversion Mutation (IVM)[134]
- Scramble Mutation (SM)[117]
- Partial Shuffle Mutation (PSM)[135]
- Reverse Sequence Mutation (RSM)[135]

The most common mutation operator for non-ordered genetic algorithms is to select a gene at random and assign a new random value to that gene from the list of possible values. However this is not a viable option for ordered genetic algorithms as this creates points that are visited more than once or not at all, which is a non-legal solution to the traveling salesman problem. Therefore special operators have been designed to prevent this from occurring. The special operators work on the basis of altering the order of chromosomes instead of the individual value of each gene. This ensures that no two points are repeated as well as no points being omitted. There are two main types of mutation operators, the first type of mutation switches the position of two or more genes and the second type of mutation reverse the order of a number of genes within a chromosome.

A more unique mutation operator was developed by Sengoku[110]. A gene is selected at random and all of the possible paths are analysed. The mutation operator then chooses any new path that is shorter then the current path. This technique increases the likelihood of a beneficial mutation which decreases the time taken to converge on a solution.

A study was performed by Abdoun which discusses and analyses a number of mutation operators for the traveling salesman problem[135]. These include: Twors mutation, Centre inverse mutation, Reverse sequence mutation, Throas mutation, Thrors mutation and Partial shuffle mutation. The various mutation operators were used in solving the BERLIN52 traveling salesman problem and then analysed to compare the performance of the various mutation operators. The study found that the Reverse sequence mutation operator outperformed the other operators by a significant amount. It cannot be said that the Reverse sequence mutation operator is the best choice for all traveling salesman problem scenarios as it was only tested on one problem.

Merz[136] revisited the most commonly used operators for solving the traveling salesman problem and identified potential improvements for these operators. It was also concluded from the study that the performance of various operators depends greatly on the rest of the structure of the genetic algorithm. One mutation operator can work very well with one crossover operator but less well when used with another crossover operator.

A more recent study performed by Al-Dulaimi developed a Travelling Salesman Problem Genetic Algorithm (TSPGA) which tested the effects of mutation rate on three commonly used crossover operators: OX, PMX and CX[137]. It was found that the mutation rate has a large impact on both the convergence rate as well as the quality of solution generated for all three operators. The OX operator was most sensitive to mutation rate whereas the PMX and CX operators were less sensitive to mutation rate.

A novel mutation operator was developed by Liu which instead of being completely random has a learning function which increases the likelihood of a beneficial mutation[138]. This new mutation operator almost consistently produced optimal solutions to many standard traveling salesman problems and outperformed the Edge Assembly Crossover algorithm with respect to reaching an optimal solution. The drawback to the complex mutation operator is the computational effort required which dramatically increased the time required to converge on an optimal solution.

An innovative genetic algorithm based on immunity and vaccines was developed by Jiao to solve the traveling salesman problem[139]. Local areas of high fitness will become immune to mutation and these areas of high fitness can also be "injected" into low fitness individuals. This technique has proven to decrease the time taken to converge on an optimal solution as well as increase the quality of solutions generated.

A hybrid genetic algorithm was developed by Gupta which uses a special heuristic to create an initial population as opposed to a completely random initial population[140]. The hybrid algorithm was tested on three separate problems and compared the performance of the hybrid algorithm to a simple genetic algorithm. It was found that the generated solutions were a lot better for the hybrid algorithm than those generated by the simple genetic algorithm.

Uğur developed a genetic algorithm which generates solutions to traveling salesman problems whose points lie on the surface of a cuboid[141]. This algorithm could be adapted to be used for machining parts with features on more than one face which will require reorientation of the part using a five-axis machine.

(ii) Ant Colony Optimisation

Ant colony optimisation is another type of heuristic problem solving technique. The idea was first proposed by Dorigo in 1991[142] which was further developed until 1996[143]. Dorigo then developed the Ant colony System (ACS) in 1997[144] which produced better results for most of the problems tested than other heuristic processes. It can be seen as a multi-agent system modelled after the pathfinding behaviour of ants in nature. As ants travel to and from locations they leave behind a pheromone along the path they travelled. This pheromone increases the likelihood of other ants following the same path. The pheromone's strength decreases over time making paths that are longer less likely to be followed. This is because the pheromones will build up faster on paths that have shorter lengths. Therefore over time the ants will eventually converge on a path that is shorter than the original path chosen by the ants.

Ant colony optimisation performs well in real time systems where obstacles can be introduced or removed at any time as the ants are very quick in altering parts of a path to account for such changes in the environment [144]. The nature of ant colony optimisation systems makes them very quick at converging on solutions for the traveling salesman problem. There have been a number of ant colony optimisation systems developed to solve the traveling salesman problem.

Ant colony optimisation has been used for path planning in many applications one of them being spray forming[145]. A study was done by Tewolde comparing the use of genetic algorithms and ant colony optimisation on path planning in spray forming. It was found that genetic algorithms produced faster results at the cost of lower quality solutions[145].

Manfrin developed an ant colony optimisation system where a number of systems were

run in parallel[146]. The various systems running in parallel communicate with each other by passing certain information between each system. Five different types of communication were tested and it was found that overall it was beneficial to run multiple systems in parallel. The developed ant colony optimisation system was tested on a large number of problems.

Tsai created a novel ant colony optimisation algorithm which included a multiple nearest neighbour and dual nearest neighbour algorithm[147]. Both methods involve adapting the algorithm which generates the initial paths. A starting point is chosen at random, and then the nearest neighbour is chosen as the next point and so on until all of the points have been visited. This algorithm has shown to dramatically improve the solutions generated as well as the time taken to converge on the optimal solution.

(iii) Simulated Annealing

Simulated annealing is a global optimisation technique inspired by the annealing process in metallurgy developed by Kirkpatrick in 1983[148]. There is very little literature relating to simulated annealing with respect to solving the traveling salesman problem. This could be due to the success of other heuristic methods.

Kirkpatrick developed a simulated annealing algorithm to solve a traveling salesman problem[149]. The problem contained a set of 6406 holes which needed to be drilled. The algorithm performed well for this problem but the simulated annealing algorithm was not compared to any other existing techniques in this study. Therefore it is hard to say whether this method performs better than other heuristics.

Bookstaber developed an algorithm to solve the traveling salesman problem using simulated annealing[150]. The algorithm was tested on a 100 city problem where the points were positioned in a uniform grid. The most optimal solution generated by the algorithm had a length of 106.142. The optimal solution has a length of 99. Therefore the accuracy of the algorithm is quite low when compared to the other heuristic processes.

Malek performed a study comparing simulated annealing to tabu search algorithm when solving the traveling salesman problem[151]. Both algorithms produced solutions to the

various traveling salesman problems tested, however the tabu search method outperformed simulated annealing for each test.

3.6.3 Multi-Objective Meta-Heuristics

Many machining optimisation problems contain more than one objective. This is especially the case with five-axis machining toolpath generation. This is due to the many variables that need to be considered in five-axis milling as the angle of the tool has to be considered as well as the position. The only efficient method of solving these types of optimisation problems is by using a multi-objective algorithm.

A novel method of defining the genes and structure of the generated tool path was developed by Car[152]. The surface of the part was converted into a 2D grid with one cell being the start of the tool path and another cell the end. From the start point, the tool path can move in one of four directions (up, right, down, left). This continues for each cell until the end point cell is reached. The fitness is then calculated as a multi-variable function of how many cells have been visited, the number of turns and the overall distance travelled. A novel crossover operator is used for this genetic algorithm as it requires the ability to handle varying lengths of chromosomes. The genetic algorithm performed better than analytical algorithms for large grid sizes but not for small grid sizes. This genetic algorithm is limited in the fact that it can only generate tool paths for uniform square grids and does not allow diagonal movement. Another problem with this algorithm is that it allows for points to be visited more than once or not at all which can lead to inefficient toolpaths.

An evolutionary algorithm was developed by Weinert to optimise toolpaths for multi-axis die and mould machining[153]. The algorithm uses a three dimensional toolpath as its input and then generates a multi-axis toolpath. The variable of the algorithm is the rotational axes of the tool and the fitness is calculated by simulating the machining of the part and detecting any collisions between the tool and part. A second simulation is run to analyse the deviation of the tool trajectory to the interpolated trajectory by using the kinematic model of the machine. A third simulation is run to calculate the tool engagement while cutting

the workpiece. Therefore the evolutionary algorithm has multiple objectives to consider and optimise which increases the computational effort required to calculate the fitness of each individual. The paper did not state any times required to generate solutions. The results showed that the developed algorithm gave a slight improvement to simple parts but had difficulty converging on a solution that fit all the objectives for more complex parts.

Kersting et al. developed a multi objective evolutionary algorithm that optimises NC paths[154]. By integrating the multi-objective properties of the SPEA 2 system and the single-objective stochastic optimiser CMA-ES a new system ICSPEA was created. The two objectives of the algorithm is to produce toolpaths that deviate as little as possible to the surface normals and to produce smooth toolpaths. The results showed that the algorithm produced smooth toolpaths but had trouble also producing toolpaths close to the surface normals which led to issues with consistent tool engagement.

Kersting further developed the multi objective evolutionary algorithm that optimises NC paths produced by CAM systems[155]. Again the objectives of the algorithm is to produce a toolpath with the least deviation of the tool orientation to the surface normals and to produce smooth toolpaths. However a hybrid evolutionary algorithm was developed which first generated optimal cutter locations with respect to surface normals and then the toolpath was optimised between the generated points. The optimised toolpaths also have to consider any tool collisions and if the machine tool can achieve the required orientations. The overall toolpath to be optimised is also separated into many sub paths to decrease the degrees of freedom for the optimisation problem and reduce the time required to produce solutions. The results from the research produced improved toolpaths with less erratic movement and a more consistent tool engagement.

Jiao proposes a new multi-objective evolutionary algorithm where a sub-population is created for each objective[156]. The various sub-populations then interact with each other to obtain a global optimum. This method displayed a greater efficiency in converging on a global optimum than other multi-objective algorithms. This novel method will be useful when creating the population structure for the genetic algorithm as the toolpath generation

problem is a multi-objective problem.

3.7 Challenges in adopting TSP for toolpath generation

A conclusion that can be made from the literature is that all of the research in solving the TSP is focused on reducing the length of the path. This makes sense as the original problem of the TSP is to reduce the length, however if the TSP is to be adapted to provide solutions for machining toolpaths then other characteristics of the path need to be taken into consideration in the optimisation algorithm. The approximation and local search algorithms were designed to reduce the length of the path and would not be appropriate in optimising for other characteristics. Meta-heuristics have the advantage of specifying the objective function in the optimisation strategy which makes them ideal for optimisation of various path characteristics. The fact that there is the possibility of optimising a machining toolpath for multiple objectives at the same time, a multi-objective meta-heuristic would be the most suitable option.

From the literature on solving the traveling salesman problem it seems that multi-objective evolutionary algorithms would be an ideal tool to use to generate and optimise CNC milling toolpaths. There has been a lot of research in solving the TSP and it seems that although they might be slower than the well established Lin-Kernighan algorithm and its adaptations, the results produced by genetic algorithms are slightly superior. Genetic algorithms also have the added benefit of having a modular structure that can be customised to solve for various objectives with only a slight adjustment in the overall architecture of the system. The approximation algorithms can be used to generate the initial population of the GA. The survey performed by Jiang of all the current optimisation algorithms for the TSP revealed that the combination of an evolutionary algorithm with a local search algorithm generated the best results in solving the TSP.

There are many tested operators for genetic algorithms which are both ordered and non-ordered as well as for single or multiple objective problems. The performance of the various operators is dependent on the problem being solved and is also very difficult to predict the

performance. Testing all of the operators found in the literature would be infeasible, therefore the top three operators for each module of the genetic algorithm will be investigated.

The algorithm designed by Car [152] can be used as a baseline for this research as it was the only example in the literature that attempted to solve the same problem as this research. The performance of this algorithm is quite limited and does not tackle many of the problems that will be covered in this research therefore the comparison can only go so far.

It can be seen in Section 3.6.2 that genetic algorithms are very efficient at reducing large problem spaces and have been used to solve similar problems. The meta-heuristic to be used in solving the travelling salesman problem will therefore be a genetic algorithm. The structure of the genetic algorithm is modular in nature which allows for a high degree of flexibility in the generation of machining toolpaths for various objectives. This will also increase the ease of development as each module can be optimised independently. There is a large set of developed operators for the various modules of the genetic algorithm. This increases the chances of obtaining the most efficient selection of operators for this problem.

3.8 Developing the Genetic Algorithm

The genetic algorithm will be developed in an object-oriented programming language. This will make full use of the modular structure of a genetic algorithm by defining the various modules and sub-modules as objects. The programming language to be used will be Java as the computational platform can be developed independent of which operating system is being used. This increases the flexibility of equipment for the developed program to be tested and used with.

The first stage of development will focus on the structure of the computational platform and the genetic algorithm. To adhere to the modular approach, the inputs and outputs of each module will have to be determined first. This will help prevent any issues when switching between modules. Once this has been done then the actual development of each module can take place. The genetic algorithm can be separated into the following main modules:

- Main Module
- Data Point Storage
- Fitness Function
- Mutation Operator
- Crossover Operator
- Selection Operator
- Population

The main reason it is beneficial to have a modular approach is the fact that modules can be easily altered or replaced without affecting the overall functionality of the program. As long as the inputs and outputs remain the same, the program will function. This modular approach will be useful when testing various genetic algorithm operators. As discussed in Chapter 3.6.2, there are many operators which have been developed for solving the travelling salesman problem using genetic algorithms. The performance of each operator varies greatly depending on the structure of the rest of the genetic algorithm. Therefore the ability to switch the various operators by replacing modules will decrease the amount of time required to optimise the genetic algorithm.

The genetic algorithm would use the data points generated from Chapter 4 as its input parameters. The data points will each be labelled with an assigned number. For example, if there are 100 data points, then they will be labelled from 0 to 99 in the order that they are entered into the genetic algorithm. These data points will then be stored and used every time the fitness of a species is calculated.

3.9 Optimising the Genetic Algorithm

There are many factors that affect the rate at which the genetic algorithm converges on a solution. To optimise the genetic algorithm to converge on an optimal solution in the least

amount of time, these factors need to be fine tuned. Firstly the various factors need to be identified. The following is a list of factors which affect the rate of convergence:

- Population Size
- Mutation Rate
- Mutation Operator
- Selection Rate
- Selection Operator
- Fitness Function
- Fitness Function Weightings
- Crossover Operator

Some of the factors listed above are dependent on other factors. For example, certain crossover operators perform better with a low mutation rate whereas other crossover operators perform better with a high mutation rate. Therefore each factor cannot be optimised individually but will need to be optimised in conjunction with the other factors it is dependent on.

Optimising the population size is difficult as a large population size greatly increases the computational power required to generate a new population, however having a large population size decreases the rate at which the search-space of the genetic algorithm reduces. This generates more optimal solutions at a cost of an increased time to converge on that solution.

As there are a large number of operators developed for genetic algorithms, it would be very time consuming to program and test every operator found in Chapter 3.6.2. Therefore only the top three overall performing operators for each module will be programmed and tested in the development of the genetic algorithm.

To identify the optimum operator for each module, each of the three operators for each module will be put through a Monte Carlo test. A Monte Carlo test was chosen as the best method for this due to the stochastic nature of a genetic algorithm. The selection, crossover and mutation operators all contain random number generation elements which can lead to an overall variation between the generated toolpaths. Therefore by using a Monte Carlo test it is possible to identify which operators are outperforming the others statistically. The number of iterations required to obtain a sufficient representation of the performance of each operator will have to be determined first. The method of determining the number of iteration required is described in a study by Mundform et al[157]. An initial test is required at a much lower number of iterations to identify the variation in the generated toolpaths. From this it can be determined how many iterations are required to be within a set confidence level.

Once the operators have been selected, a sensitivity analysis will have to be performed on the other remaining factors. This will determine the sensitivity of the various factors and their dependency on each other. This information will be vital in the optimisation of the genetic algorithm. The sensitivity analysis will be performed as described by Iooss et al[158]. The results of the analysis will, in turn, be used to fine tune the parameters of the algorithm.

3.10 Validation

The various models developed in this research will need to be validated. One way to validate the results of a meta-heuristic is to compare the results generated by the meta-heuristic to results generated by solving the same problem by brute force. Due to the nature of the problem being solved, it will be difficult to fully validate a large sized travelling salesman problem by brute force within the time constraints of the research as well as the equipment available. However there is a database of standard travelling salesman problems which also have a global optimum solution for testing purposes[159]. This library of sample instances of the travelling salesman problem is known as TSPLIB. The library contains a variety of travelling salesman problem types including the symmetrical travelling salesman problem which most closely resembles the problem being solved in this work.

The global optimums of the TSPLIB only take into account the path length when solving the TSP. Therefore these global optimums can only be used to verify the first objective function of the genetic algorithm. The nature of the points in the standard problem sets of the TSPLIB vary slightly from the TSPs being solved in this research. The point sets generated for the toolpath problem will produce a uniform grid as opposed to the TSPLIB problems which contain a non-uniform set of points. Fundamentally the nature of the problem being solved is the same as the position of the grid points do not influence the method of which the problem is solved if the path length is the only objective being optimised.

The other objective functions will have to be validated by comparing the generated paths to the global optimums identified by the brute force method. This will result in the validated problems containing fewer points than there would be in a normal toolpath generation problem. Therefore a method has been devised which compares the generated results to the brute force results for a set of problems with incrementing number of points to verify that the number of points in the problem does not influence the quality of the generated results from the genetic algorithm.

Chapter 4

Creating a Computational Platform for Generating Optimised Toolpaths

This chapter discusses modelling toolpath generation in the form of a travelling salesman problem.

4.1 Theoretical Model of Toolpath Generation as a Travelling Salesman Problem

The main objective of this research as outlined in Chapter 1.2 is to generate and optimise a machining toolpath for a given feature. The various techniques to do this were discussed in Chapter 2, however one method that has not been explored is to model this problem as a travelling salesman problem. This is due to the large solution space that exists when solving the travelling salesman problem.

If the problem can be simplified sufficiently without losing too much accuracy then the travelling salesman problem becomes easier to solve. By coupling the problem simplification with a suitable optimisation heuristic which is designed to solve the adapted travelling

salesman problem it will be possible to find a solution.

The number of solutions that exist for a traditional travelling salesman problem is entirely dependant on the number of points in the problem set. As any point can connect to any other point and each point has to be visited at least once, the number of solutions can be calculated as follows:

$$Number of Solutions = N! \tag{4.1}$$

Therefore reducing the number of points in the problem is the only way to simplify the problem. Theoretically there are an infinite number of positions that a tool can occupy in a toolpath, but the available positions that the tool can occupy can be limited in such a way that a valid toolpath can still be created from the set of points.

Before the problem can be solved it is important to set up the problem in the most efficient way to make it as easy as possible to solve whilst still giving a useful solution. To do this the minimum number of points to represent a machining volume needs to be determined. The machining volume can be modelled as a three dimensional grid with an appropriate resolution. The maximum spacing between points needs to be calculated in order to reduce the number of points in the grid as much as possible. This will be dependent on various factors such as the accuracy required, feature shape and tool parameters.

Once the grid has been set up, the custom travelling salesman problem can be developed. The mathematical model for traditional travelling salesman problems can be seen in Chapter 3.5 with Equation 3.2. This model will have to be adjusted to solve the problem of toolpath generation. The $d_{i,j}$ term in Equation 3.2 represents the cost of going between two points in the grid. The cost can be altered in such a way as to optimise different parameters of a path.

For this research, three main attributes of toolpaths will be analysed and optimised. The overall length of the path, the average tool engagement of the path, and the straightness of the path. The path length is important as the longer a toolpath is, the longer it takes to machine a feature or part. Therefore by reducing the path length, more parts can be produced in the same amount of time. To use path distance as a measure of cost, the cost

term in Equation 3.2 will be replaced with Equation 4.2.

$$d_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{4.2}$$

Where the x, y, and z terms are the axis coordinates of each point. This equation can be used for a three dimensional grid. For a two dimensional grid the same equation can be used with the z term removed or set to zero.

The tool engagement is important as the amount of tool engagement can have an effect on the tool life and the quality of finish of the part. A constant tool engagement will produce a better surface finish of the part due to the chips being produced by the tool will be the same size and this will leave a smoother surface. By reducing the tool engagement, the force applied to the tool is reduced and this will reduce the wearing of the tool and increase the tool life. A lower tool engagement also results in a longer machining time as the volume of material being removed is reduced and it will therefore take longer to remove all of the required material. It is then important to ensure that the tool engagement is below a specified threshold but as close as possible to that threshold throughout the entire toolpath. To optimise a toolpath with respect to the tool engagement, the cost term in Equation 3.2 will be replaced with Equation 4.3

$$d_{i,j} = \frac{T_{i,j}}{N} \tag{4.3}$$

Where $T_{i,j}$ is the average tool engagement between points i and j and N is the number of points in the toolpath. This will give the average tool engagement for all of the sub-paths in the toolpath when substituted into Equation 3.2. Then instead of optimising for the minimum value in Equation 3.2 the optimum will be getting the output of Equation 3.2 to be as close as possible to a given value of tool engagement. A consistent tool engagement is a beneficial characteristic for a toolpath to have as it ensures that the chips being formed by the tool will be of constant size. This leads to a better surface finish and less tool wear.

It is important to keep a toolpath as straight as possible which can be done by avoiding

sharp turns. This is because sharp turns in a toolpath can lead to kinematic errors in the machine due to high acceleration of the motors. The straightness of a toolpath can be calculated either by the sum of the angles between points, or as a number of non straight sub-paths in the toolpath. The resulting toolpath will vary depending on which type of straightness is measured. Both methods of optimising for straightness will be modelled and analysed to investigate their effects on the generated toolpaths.

The optimisation of the straightness of the toolpath will be a secondary goal of the optimisation algorithm. This is because it is important to keep the toolpath straight regardless of whether the primary goal is toolpath length or tool engagement. Therefore the optimisation algorithm will be a multi-objective algorithm. Having multiple objectives can lead to a larger solution space and an increased complexity in defining the optimisation algorithm. However, when properly balanced and tuned the optimisation algorithm can produce the required results. The development and balancing of the objective function can be seen in Section 4.3.3 of Chapter 4.3.

4.2 Creating a model from STEP-NC Data

This section explains the process of generating an appropriate model of a part or a feature with the purpose of creating a machining toolpath to manufacture said part or feature. The design of the model has several requirements for it to thoroughly fulfil the goal stated above. The requirements are as follows:

1. The model should be able to handle features described by the STEP-NC (ISO14649) standard.

2. The model should allow for machining toolpaths to be generated solely from the data contained within the model.

To satisfy the first requirement, the model will have to be able to store three dimensional information about the part or feature. This will allow 2.5D and 3D features to be modelled. Also the model should be precise enough to retain complex surface and boundary information

which will allow accurate toolpaths to be generated from the model. A process is also required that will interpret the STEP-NC data to be able to generate a model from.

To satisfy the second requirement, a data storage module will be created to hold any necessary information needed to generate a toolpath from. The minimum amount of information needed to generate a toolpath will be analysed. The module will also allow additional non-essential data to be stored to create more comprehensive toolpaths. Safeguards will be put into place to ensure all essential information is available in the STEP-NC code before generating the model.

The process for generating a model from STEP-NC code will be split into several sub-processes. The first step is to parse through and interpret the STEP-NC code. All essential and non-essential information will be extracted and stored in a Java data storage. The feature and part boundaries need to be converted into polygons. The boundaries need to be adjusted appropriately so that the areas contained within the new boundaries represent all the legal positions for a machining tool to occupy. Finally a three dimensional array of points will then be generated to represent the material that needs to be removed from the raw billet to create the finished part. The model now contains all of the required information to create a machining toolpath to manufacture a specified part.

The following sections will describe each step previously mentioned in detail.

4.2.1 Interpreting the STEP-NC Code

Before developing a STEP-NC interpreter, it is important to investigate what information is required to generate a toolpath for a feature. STEP-NC code can be rich in manufacturing information and part information but only a portion of that information is actually relevant. To generate a toolpath for a feature, the minimum amount of information required is the boundary and depth of the feature and the size of the tool. This allows the volume of material that needs to be machined to be calculated. Additional information can be used to create a more specific toolpath such as a machining strategy or an average tool engagement. Therefore the list of information that will be analysed by the STEP-NC interpreter will be

split into two categories, essential and non-essential information.

Essential Information Any information that is required to create a toolpath which will produce a finished feature within the specifications of a part will be classified as essential information. The following is a list of essential information:

- Tool radius
- Feature boundary
- Boss boundaries
- Feature depth
- Feature wall slope

Non-Essential Information Any information that adds complexity or detail to the generation of machining toolpaths will be classified as non-essential information. This information can generate more favourable toolpaths or increase the performance of the overall machining for the part. The following is a list of non-essential information:

- Cut depth
- Tool engagement
- Safety height
- Cutting parameters

Now that the information to be extracted has been identified, a parser can be created that will parse through all of the STEP-NC data, extract the necessary data and store the data in the storage module. This can be developed in Java as a STEP-NC translator has been developed by Aydin Nassehi which converts the STEP-NC code from an Express data structure to a Java data structure. This allows for easier access to the STEP-NC data with a Java program. Each instance in the part 21 code can be created into a Java object. All of the

instances can then be stored in an array which is called the population. Then the interpreter can iterate through all of the objects in the population array until any objects containing essential or non-essential information is found. The information can then be transferred into the data storage module to be later used in creating the part or feature model.

4.2.2 Extracting Boundary Information

To generate a basic toolpath a model of the machining area needs to be created. Feature boundaries can be modelled in various ways within the STEP-NC data structure. All boundary profiles in STEP-NC are created by using a set of points with either straight lines between the points or a type of curve. A set of points with straight lines between them is called a polyline. This is the simplest form of boundary representation and the most straightforward to convert from the STEP-NC model to a Java model. One method of modelling curved boundaries is by using a composite curve. This method uses a segment of a circle to model a curved path between points. This is done by using the circle centre point, radius, start point and end point to draw a curved path between the start and end point. An example of a composite curve and how it is formed can be seen in Figure 4-1.

Instead of developing a program to deal with all of the different types of curve representations in STEP-NC, only the composite curve representation was considered. This is due to the fact that a program was developed by Xianzhi Zhang [160] which could interpret G-Code and convert it into STEP-NC using only polyline and composite curve segments to represent the feature boundaries. As these two types of boundary segments could accurately model any generic feature boundary, it was decided that it was not necessary to account for the various other curve types.

To convert the boundary information into a usable data structure, the STEP-NC boundary segments are converted into the Java data structure by parsing through the STEP-NC boundary data points and instantiating equivalent Java objects to store the boundary information in. Polyline segments can easily to converted as the coordinate information will be the same in either data format, however composite curve segments need to be approximated

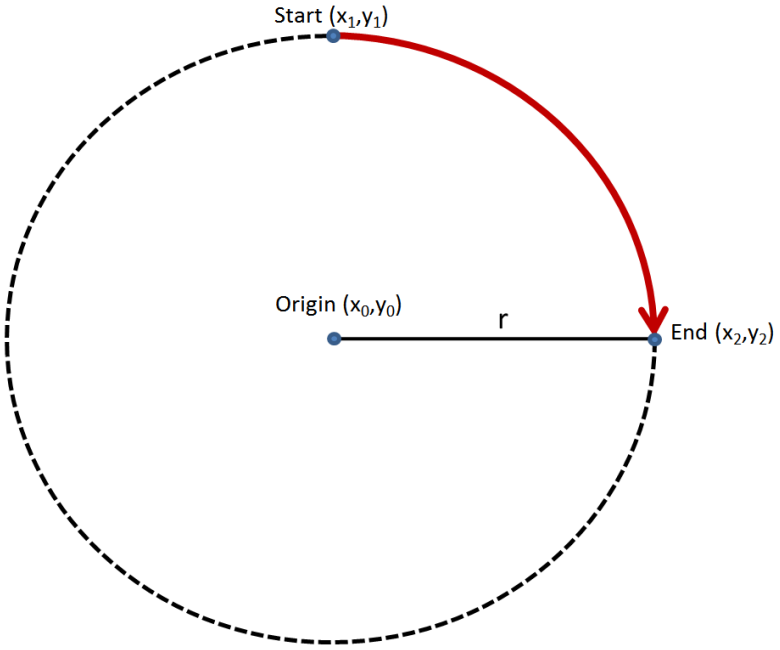


Figure 4-1: The construction of a composite curve.

to a polyline for any useful operations to be performed.

To approximate the composite curve segment as a polyline, the circular path needs to be sampled into a number of straight line paths. To make sure that accuracy is not lost the sampling rate has to be adjusted accordingly. Longer circular paths need to be sampled more frequently than short paths to maintain a similar accuracy. The error in approximating a curve to a set of straight lines is dependant on the number of points used. Figure 4-2 illustrates the error that occurs in approximating a curve.

Figure 4-3 demonstrates how this error can be reduced by inserting additional points into the polyline. The error between the straight line and curve can be calculated as long as the radius of the curve and the angle between two points in the polyline. The radius of the circle and the start and end points of the curve are all given in the STEP-NC object for a composite curve. The angle between consecutive points in the polyline depends on the number of points contained within the polyline. Therefore it is possible to calculate the number of points needed to attain a specific value of error. Inversely you can calculate the approximation error with a specific number of points. This is useful as the more points that are contained within

the polyline, the more computations that will be required in the following steps in creating the model. Therefore having the least number of points possible will speed up the model creation process.

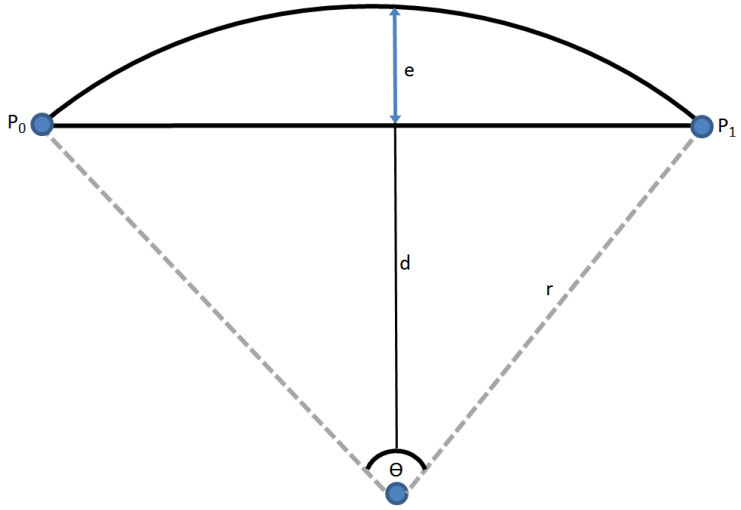


Figure 4-2: The error in approximating a composite curve.

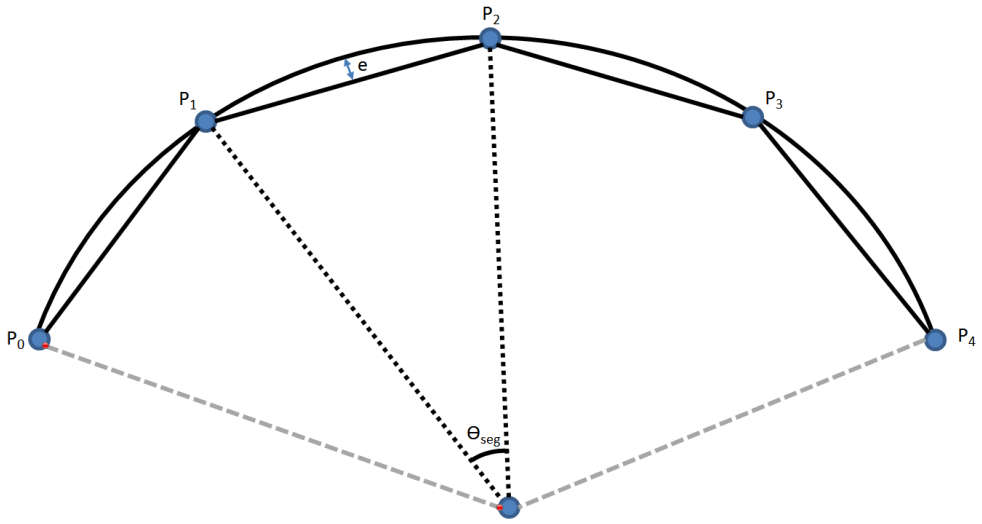


Figure 4-3: The reduction of error in approximating a composite curve.

From figure 4-2 the relationship between the segment angle and the approximation error as described by Equation 4.4 can be observed.

$$\cos\frac{\theta_{seg}}{2} = \frac{d}{r} \tag{4.4}$$

The segment angle can be calculated by dividing the total angle by the number of lines as seen in Equation 4.5.

$$\theta_{seg} = \frac{\theta}{P-1} \tag{4.5}$$

By substituting the segment angle from Equation 4.5 into Equation 4.4 and solving for the number of points P it is possible to calculate the minimum number of points required to achieve an error of e as seen in Equation 4.6.

$$P = \frac{\theta}{2\cos^{-1}\frac{r-e}{r}} + 1 \tag{4.6}$$

By using Equation 4.6, composite curves can now be approximated to a polyline with sufficient accuracy as specified by the user. This will allow all of the boundaries detailed within the STEP-NC code to be modelled and stored within the Java data structure.

4.2.3 Offsetting the Boundary

The current boundaries represent the area of material that needs to be removed. However, what is required for the final model is the bounded area of all the legal positions that a machine tool of specified diameter can located. Therefore all of the boundaries need to be offset by the tool radius so that at no position within the offset area, the tool area will not cross the original boundary. Figure 4-4 demonstrates how a tool position within the original boundary can allow the tool area to cross the feature boundary.

To avoid this, all of the edges present in the various boundaries need to be offset by the tool radius. Feature boundaries will be deflated and boss boundaries will be inflated to ensure the tool area will always be within or on the specified boundaries. Figure 4-5 shows an example of how the feature boundary offset works.

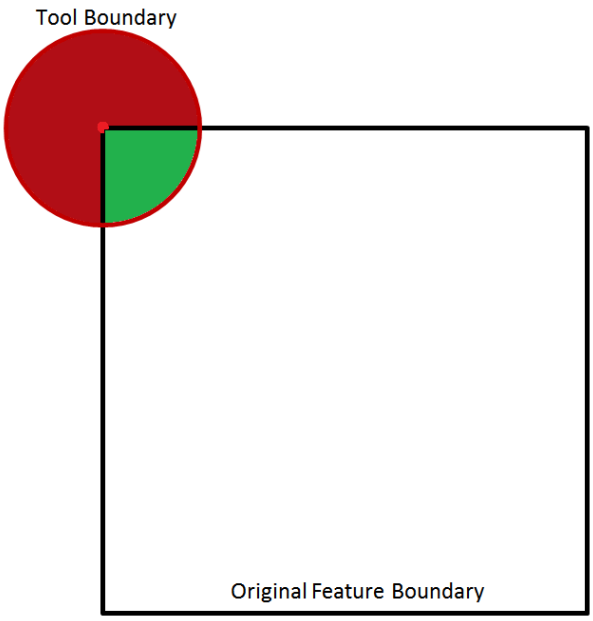


Figure 4-4: An illegal position of the tool within the original boundary.

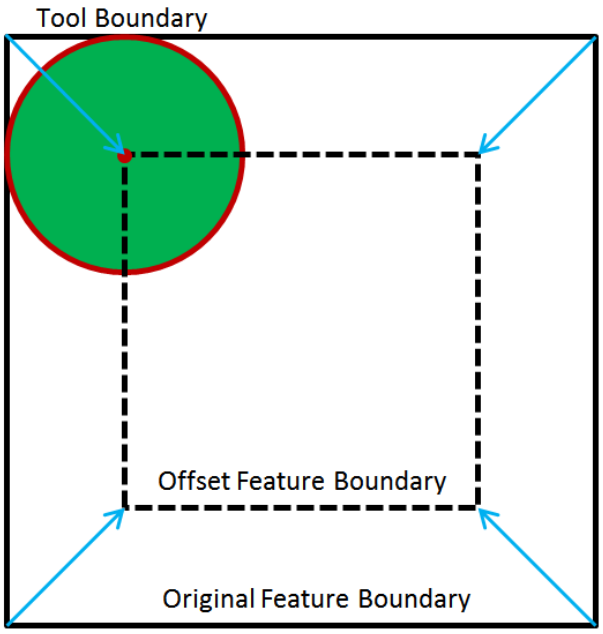


Figure 4-5: Offsetting the original boundary to remove illegal positions.

Before the boundary can be offset, the direction of the boundary must be identified. This is to determine which side of the boundary contains the material to be removed. This can be done by using Gauss's area formula (also known as the Shoestring formula)[161]. Gauss's area formula can be seen in Equation 4.7.

$$A = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right| \tag{4.7}$$

Gauss's area formula subtracts the area inside the polygon from the area surrounding the polygon. If the result is negative than the polygon is clockwise and the inside of the polygon will follow the right side of the boundary. Therefore in the case of a clockwise polygon, each edge will be offset towards the right.

The next step is to identify whether a vertex has a reflex angle. A regular vertex will only require one point to model the offset vertex whereas a reflex vertex will require two points. This will ensure that the original boundary will be formed when a tool traces the offset boundary. Now all of the edges of the boundary will be offset to their right by the tool radius. An example of this process can be seen in Figure 4-6. The green line represents a new edge created from a reflex vertex.

Once all of the edges have been offset, the area inside the new polygon represents all of the legal positions for the tool to occupy. This area will now need to be converted into a discreet set of grid points so that a travelling salesman problem can be formed from the points and then be solved.

4.2.4 Generating a Z-Map

From the literature it was found that an efficient method of representing the machining area is by using a Z-map. As it is now possible to extract the machining boundaries from the STEP-NC code, a Z-map of the machining area can be created.

The first step is to assess the maximum and minimum points in all of the axes to obtain a complete envelope of all possible locations for a grid point. This is done by parsing through the boundary points and identifying the maximum and minimum coordinates. Then an evenly

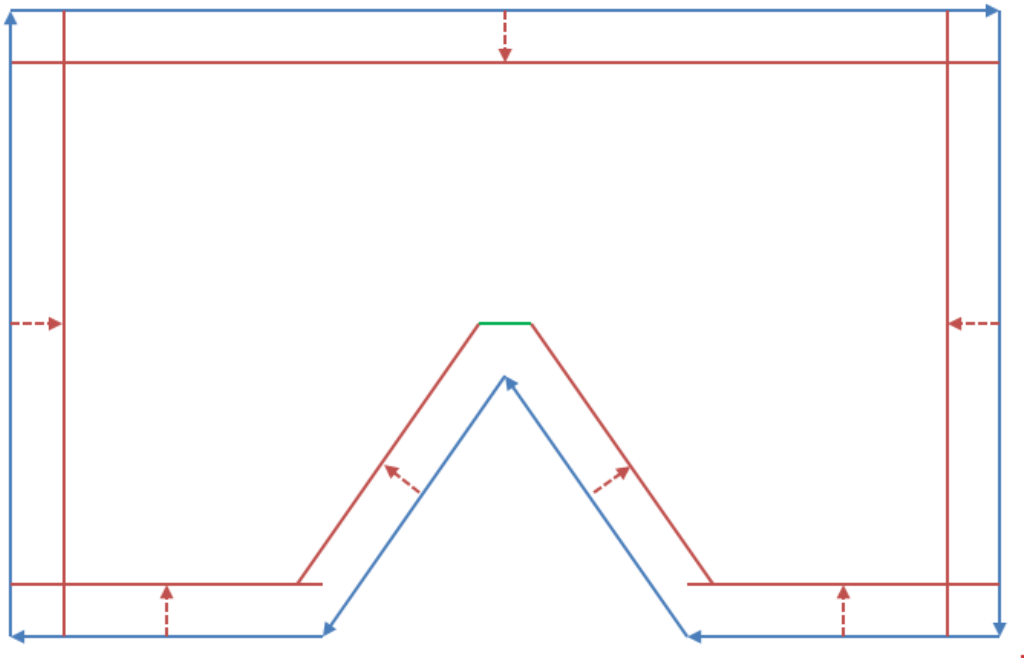


Figure 4-6: An example boundary containing a reflex vertex with all of its edges offset

spaced grid of points will be generated between the minimum and maximum points. The spacing of the grid depends on the tool diameter and the tool engagement if available. If no tool engagement value is available then the grid spacing will simply be the tool diameter. As this is the maximum allowed spacing between points where if all of the points are visited, all of the required material will still be removed. It is important to reduce the number of points in the grid as much as possible as the computing power required to generate toolpaths is almost solely dependent on the number of points in the grid. If a value of the tool engagement is available then the grid spacing will simply be as follows:

$$GridSpacing = ToolDiameter \times ToolEngagement \tag{4.8}$$

This requires the tool engagement to be given as a percentage. The tool engagement is a metric which describes the amount of material that is touching the leading edge of a tool as a percentage of the total area of the leading edge of the tool.

Once all of the grid points have been generated, they have to be tested to see which points lie within the feature boundaries and outside of any boss boundaries. To do this the ray

casting method developed by Taylor[162] was used. This method casts a line both forwards and backwards along the x axis from the point that is being tested. It then calculates the number of boundary lines that are crossed to assess whether the point is within the specified boundary or not. An illustration of the ray casting method can be seen in Figure 4-7. The number of boundary crossings from the given point to either the maximum or minimum value on the x axis. If the number is odd then the point being tested is inside of the boundary and if the number is even then the point is outside the boundary.

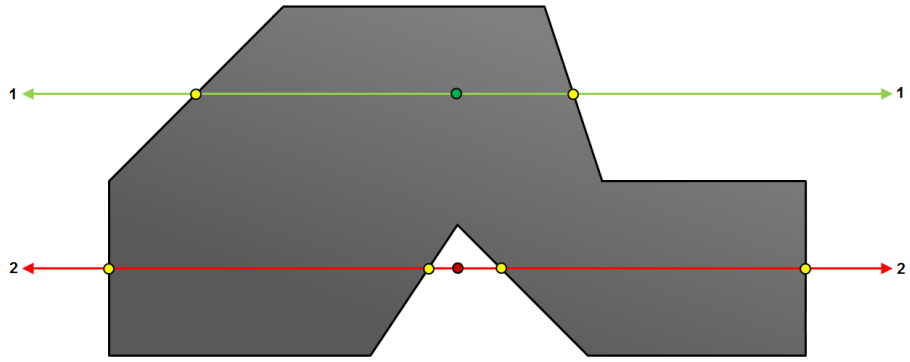


Figure 4-7: An example of the ray cast method with a point inside and a point outside the boundary.

As this method only checks whether a point is within a boundary and not on a boundary, a separate operation has to be performed to see whether the point being tested intersects with any of the boundary lines for the specified boundary. Therefore any point that lies on a boundary or within a boundary will be accepted and added to the set of grid points. This process of filtering through the grid points can be seen in Figure 4-8.

If the feature contains a boss then a similar check will have to be performed. For a boss boundary check, a grid point will be accepted if the point lies on the boundary or outside of the boundary. This will be done for all of the boss boundaries present in the feature if the feature contains more than one boundary.

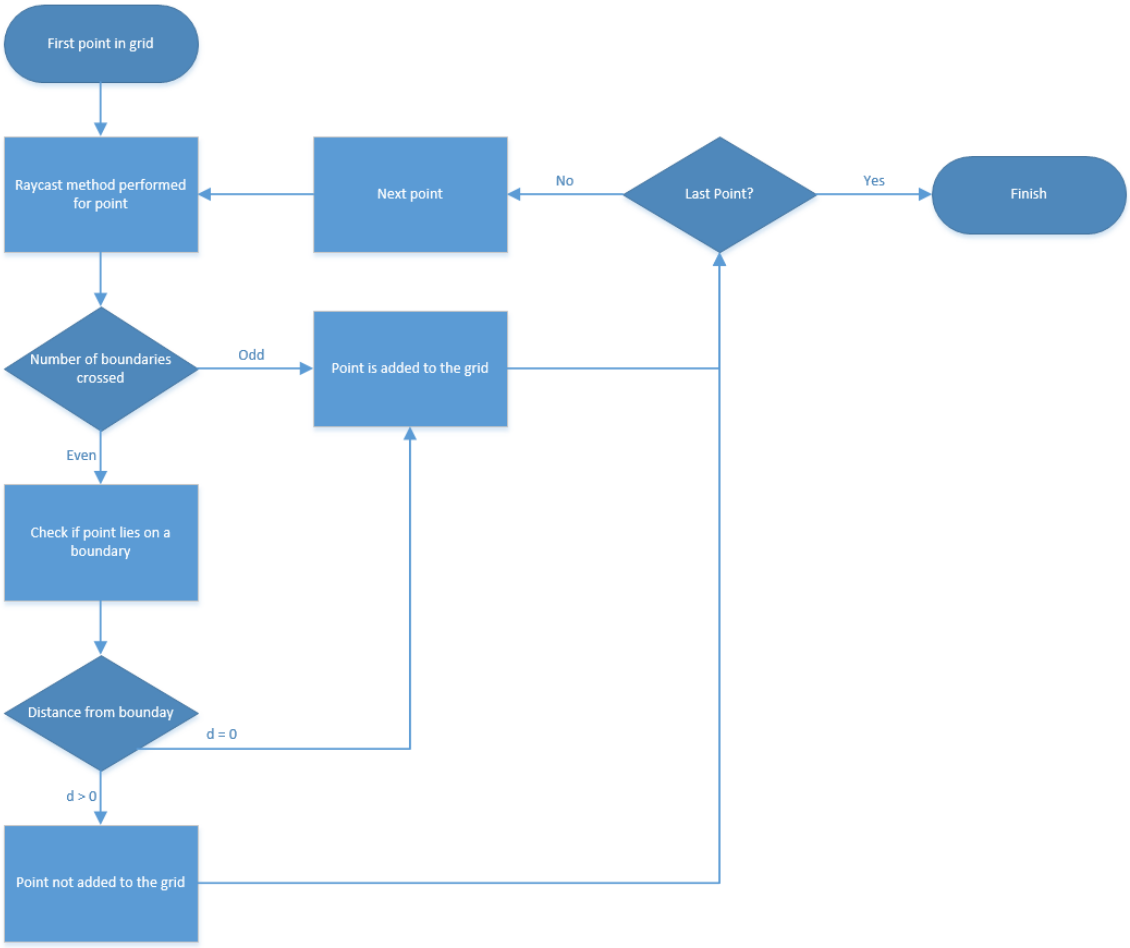


Figure 4-8: Grid point selection flowchart.

4.2.5 Generating a Z-Map for a 3D Feature

For 2.5D features, the previously described process will work and each grid of points will be identical for each cut depth. This is because the cross section of the machining area for the feature remains constant regardless of depth. However, 3D features can have sloped edges or planes with varying depths. This means that the machining area will vary depending on the depth and a new grid of points will have to be created for each depth.

The depth of each cut will depend on the specifications of the tool as well as the tolerance of the feature. As the machining volume will be cut into layers, there will be an error in the machining due to the scallops left between the various cut depths. An example of the scallops can be seen in Figure 4-9.

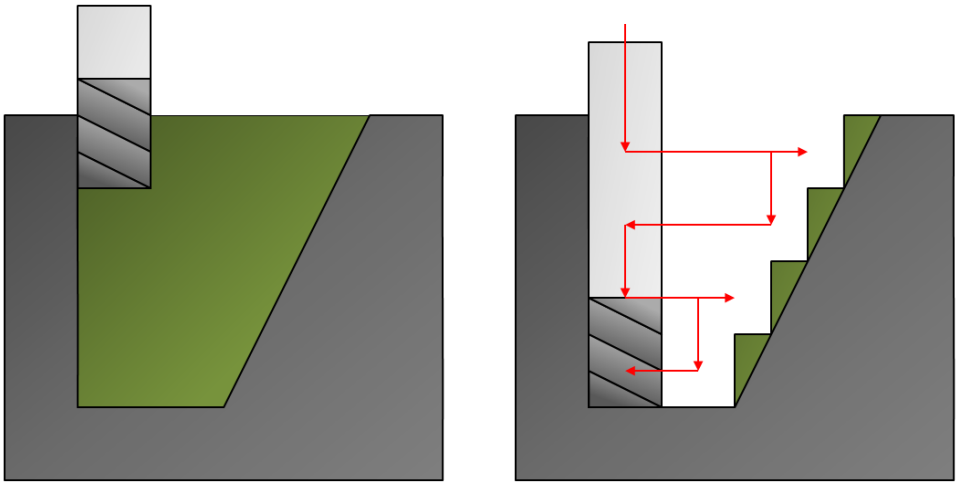


Figure 4-9: An example sloped pocket with scallop error between layers.

This error can be reduced by decreasing the cut depth of the tool. This will result in an increased number of layers for the program to process. Therefore the smaller the scallop error, the longer it will take to generate a toolpath for the feature, as well as the longer it will take to machine as there is an increased number of passes. Figure 4-10 shows a reduction in the scallop error when compared to Figure 4-9 but at the cost of having twice as many passes.

Once the cut depth has been selected, the machining area for each layer needs to be

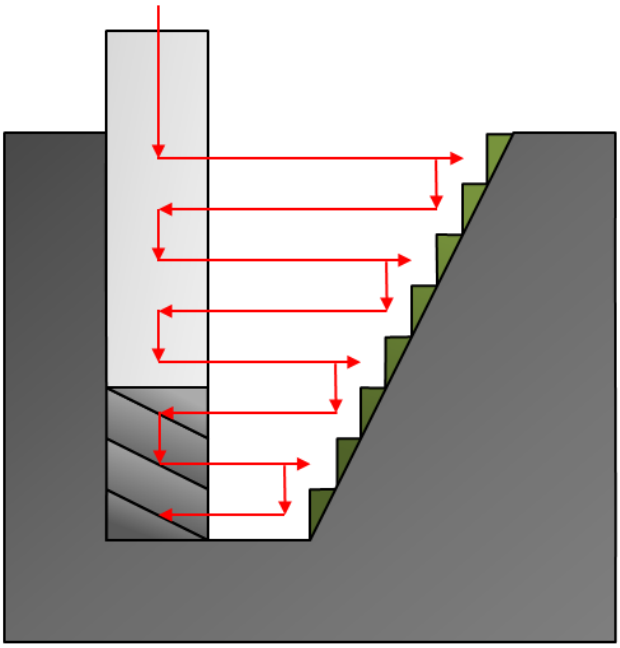


Figure 4-10: A reduction in the scallop error between layers by using a smaller cut depth.

calculated. This is done by offsetting the sloped boundaries by the appropriate amount for each layer. The offset will depend on the slope angle and the cut depth. The slope offset calculation can be seen in Equation 4.9.

$$Slope\ Offset = Cut\ Depth \times \tan\theta \tag{4.9}$$

Where θ is the slope angle specified by the STEP-NC program. The slope offset will be added to the tool radius offset to calculate the total offset required by the program to adjust the machining area to account for the slope. The total offset calculation can be seen in Equation 4.10.

$$Total\ Offset = Tool\ Radius + (Cut\ Depth \times \tan\theta) \tag{4.10}$$

The optimisation algorithm will have to run for each layer as the machining area will vary slightly between the layers. This will require a lot more computational effort when compared to a 2.5D feature but will result in a more accurate feature as this method can now account

for sloping boundaries. The higher the accuracy that is needed from the user will result in more layers being generated and thus more computational effort.

The Java program can now successfully read through STEP-NC data, extract the necessary information required to make the model, convert boundary information into Java polyline objects, and offset the boundaries appropriately depending on various factors. These steps can be visualised in Figure 4-11a.

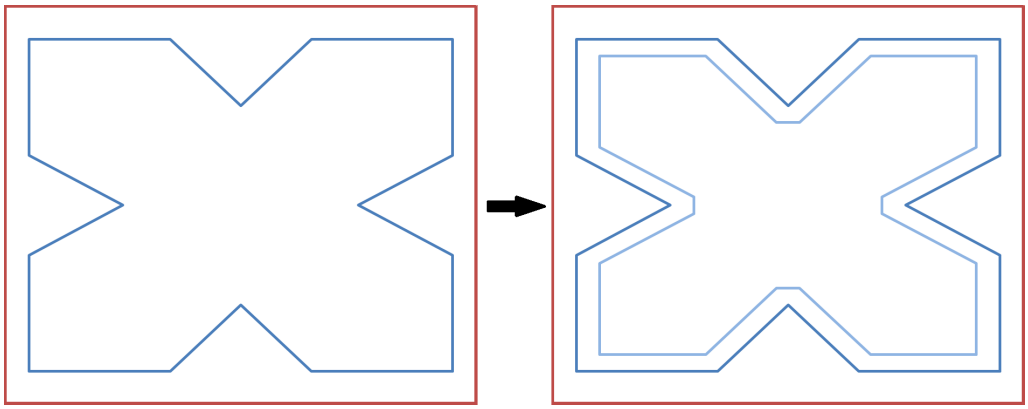
The program can also generate a uniform grid of points inside the area of the specified boundaries accounting for any boss boundaries. This step can be seen in Figure 4-11b. All of the green points are accepted grid points and any red grid points lie outside of the offset feature boundary.

Now that the grid of points has been generated for the machining area of the feature, any path that goes through all of the points without intersecting a boundary will be a valid toolpath as all of the material will be machined. To generate an optimal toolpath will require some sort of optimisation process which will be explained in the following section.

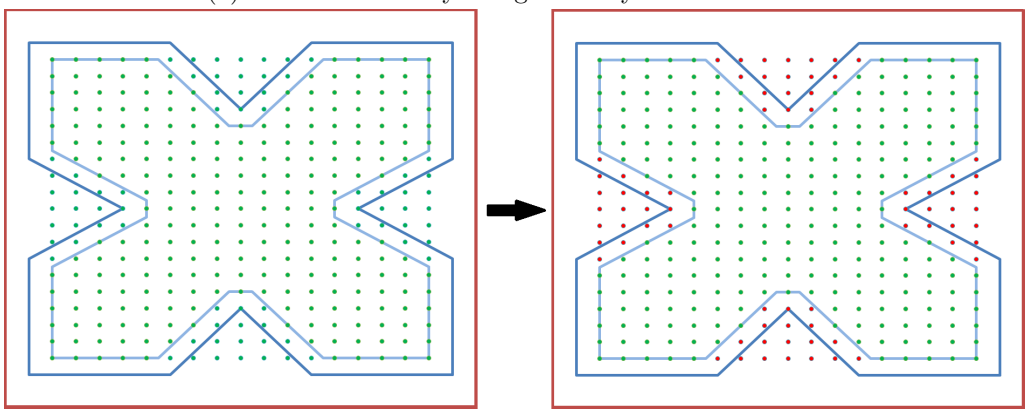
4.3 Developing the Genetic Algorithm

After reviewing the literature and researching the various optimisation algorithms that have been developed, it was decided to use genetic algorithms as the heuristic tool to solve the toolpath generation and optimisation problem. This was because of their modular structure and flexibility with respect to defining various objective functions and being able to easily switch between them. Genetic algorithms are very efficient at reducing the solution space quickly and finding near optimal solutions to complex problems.

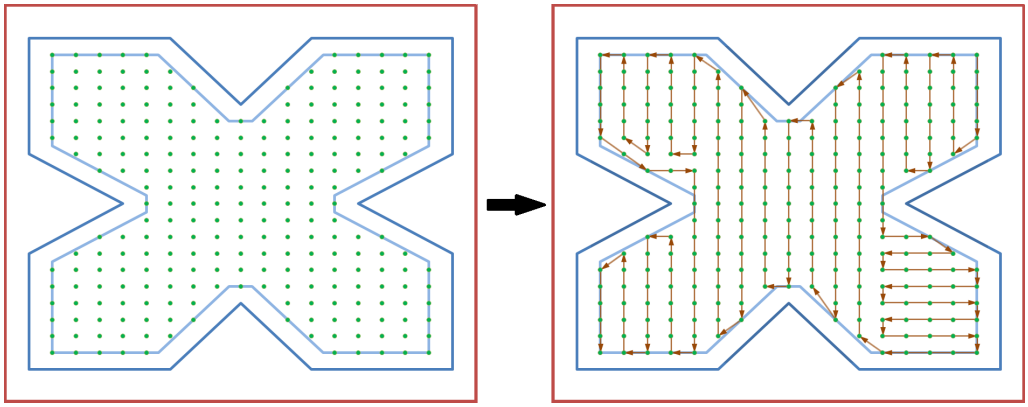
The travelling salesman problem is a very complex problem with a very large solution space even with a relatively low amount of points to travel between. The toolpath generation problem outlined in Chapter 4 is a multi-objective optimisation problem which creates an even more complex solution space and therefore genetic algorithms are a great tool to use to solve this problem.



(a) Feature boundary being offset by the tool radius.



(b) Generated grid and the selected points to be used in the genetic algorithm.



(c) Generation of the machining toolpath.

Figure 4-11: Generating a machining toolpath for an example feature

4.3.1 Overall Structure of the Genetic Algorithm

Genetic algorithms are modelled after the process of evolution by natural selection. Instead of having a population of organisms there is a population of solutions to a problem. The population is then tested to see how well they solve the problem that has been assigned and gives each solution a score or "fitness." Then certain solutions will be selected to produce new solutions from the data they contain. Their genes are crossed over in a specific way and then mutated to form new solutions and the cycle starts again. As the generations of solutions go by, the better solutions will survive while the poor performing solutions are "killed" off and over time the solutions will evolve by better solving the objective. Eventually the process will stop improving when either a local or global optimum is found.

The genetic algorithm consists of four main modules: Selection, Crossover, Mutation, and Fitness Function. Each module has an important role to play in the optimisation process. Each module is modelled after its equivalent process in the natural mechanism of evolution. All the modules have to me carefully designed so that they all work together to achieve the same end objective.

The selection module decides which individuals are chosen to produce new offspring. The crossover module creates a new offspring by selecting and mixing the genes from the parents into the new offspring. The mutation module introduces new genetic material by randomly altering the offspring's genes. The fitness function scores how well a individual solves the problem assigned to it.

There were some drawbacks to genetic algorithms that had to be overcome. Genetic algorithms are prone to premature convergence when a solution is found that is a local minimum or maximum. Also, for non regular feature shapes and features with a boss, not all paths that can be made will be legal toolpaths as they may cross a boundary and affect the finish of the machining feature. Another problem is that for optimisation problems without a known end point it is impossible to know whether the generated solution is the global optimum or a local optimum. These three issues were taken into account throughout the development of the genetic algorithm and the means by which they were overcome are

discussed in their relevant sections in this chapter.

4.3.2 Initial Conditions of the Genetic Algorithm

The design of the initial state of the genetic algorithm is an important step as it sets up the initial solutions which will subsequently be optimised to the final solution. Before any individuals are generated, a distance matrix is created to store all of the distance values between points. This optimises the genetic algorithm by only having each distance calculated once and not repeatedly. The genetic algorithm can go through thousands of generations and with the path slowly optimising through the generations it will result in many of the path sections being repeated through this process. Therefore by using a distance matrix to store the pre-calculated distances, a lot of computing effort will be saved.

Using the distance matrix, a second nearest neighbours matrix can be created. The nearest neighbours matrix contains all of the x nearest points to each point in the grid. This matrix is crucial for generating the initial population and for the mutation operator. While generating the nearest neighbours matrix, all of the illegal paths are not considered so as to minimise the chances or any illegal toolpaths being generated. It was decided to have the 24 nearest points to each grid point be stored in the nearest neighbours matrix as this would include two layers of surrounding points to be stored which allows for more flexibility in the generated toolpath in complex sections of a feature.

If the tool engagement is to be calculated in the genetic algorithm, a separate engagement grid is generated for the feature. This grid will be a much higher resolution version of the grid already generated for the feature and will be used to track the material removed along a generated toolpath and is required for calculating the tool engagement for the toolpath. The engagement grid is essentially a bitmap which models the status of material at a certain point in the feature. Figure 4-12 is an example of a ternary grid for the example part in Figure 3-2. A value of 0 represents no material at that point, a value of 1 represents material that should be removed and a value of 2 represents material that should not be removed such as material outside of the boundaries of the feature. If the tool removes a point which had a

value of 2 it would indicate that the toolpath is illegal and will then be removed.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0
0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0
0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0
0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0
0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-12: An example ternary grid for a simple part.

To generate a new initial individual, random starting point is chosen. The next point will then be randomly chosen from the nearest neighbours matrix for that point. This process then repeats for all subsequent points while checking to make sure a point has not already been chosen. If all of the nearest neighbours have already been used for a point then a random point will be chosen for the next point ensuring that it is not an illegal move. If no legal moves remain for a certain point then the individual is scrapped and the process is restarted until a full legal path has been generated.

It is crucial that enough variation exists between the initial solutions so that there is a better chance of obtaining the optimal genetic information. Each gene contains a point in the toolpath and each individual will contain all of the possible points in its set of genes. Therefore it is not the genes itself that require the variation as is the case in most genetic algorithms but the order of consecutive genes that need to vary. The connection between two genes or points is called an edge, in the same way that two connected vertices create an edge. So to increase the variation within the initial population, the number of duplicate

edges within that population need to be as low as possible. This can be done by analysing the similarity between a new individual and the individuals already present in the initial population.

To analyse the similarity between two individuals, a list of the edges present in the new individual needs to be made and compared to the list of edges in the already existing population. If an individual contains more than 20% of the edges contained within another individual then it will be replaced by a new individual.

The population size of the genetic algorithm is a complex variable to establish. If the population size is too small, there will not be enough genetic information for the algorithm to work with and can lead to a great quantity of duplicate edges in the population. This increases the probability of early convergence in the optimisation and can result in non-optimal solutions being generated as the final toolpath. However having fewer individuals in the population decreases the computational effort required to generate each new population.

A population size of 25 was chosen for the genetic algorithm which produced a good balance between the algorithm's convergence rate and quality of generated toolpaths. Higher numbers of population were tested which did not result in better toolpaths being generated and as can be seen from Figure 4-14 a population of 25 was the lowest population possible without compromising on solution quality.

4.3.3 Genetic Algorithm Operators

Once the genetic algorithm has been set up with all of the parameters and an initial population, the algorithm now enters the main loop of the program. The program loops through all of algorithms operators until an end clause is satisfied. Ordinarily a genetic algorithm will continue optimising the population until a solution has been found which meets a certain requirement. For example, a toolpath which has a path length that is less than x. This is possible with a problem where the solution is known but the variables are not. However with generating toolpaths, it is not known what the length of an optimal toolpath will be for a feature. For this reason, the genetic algorithm will continue looping until a number

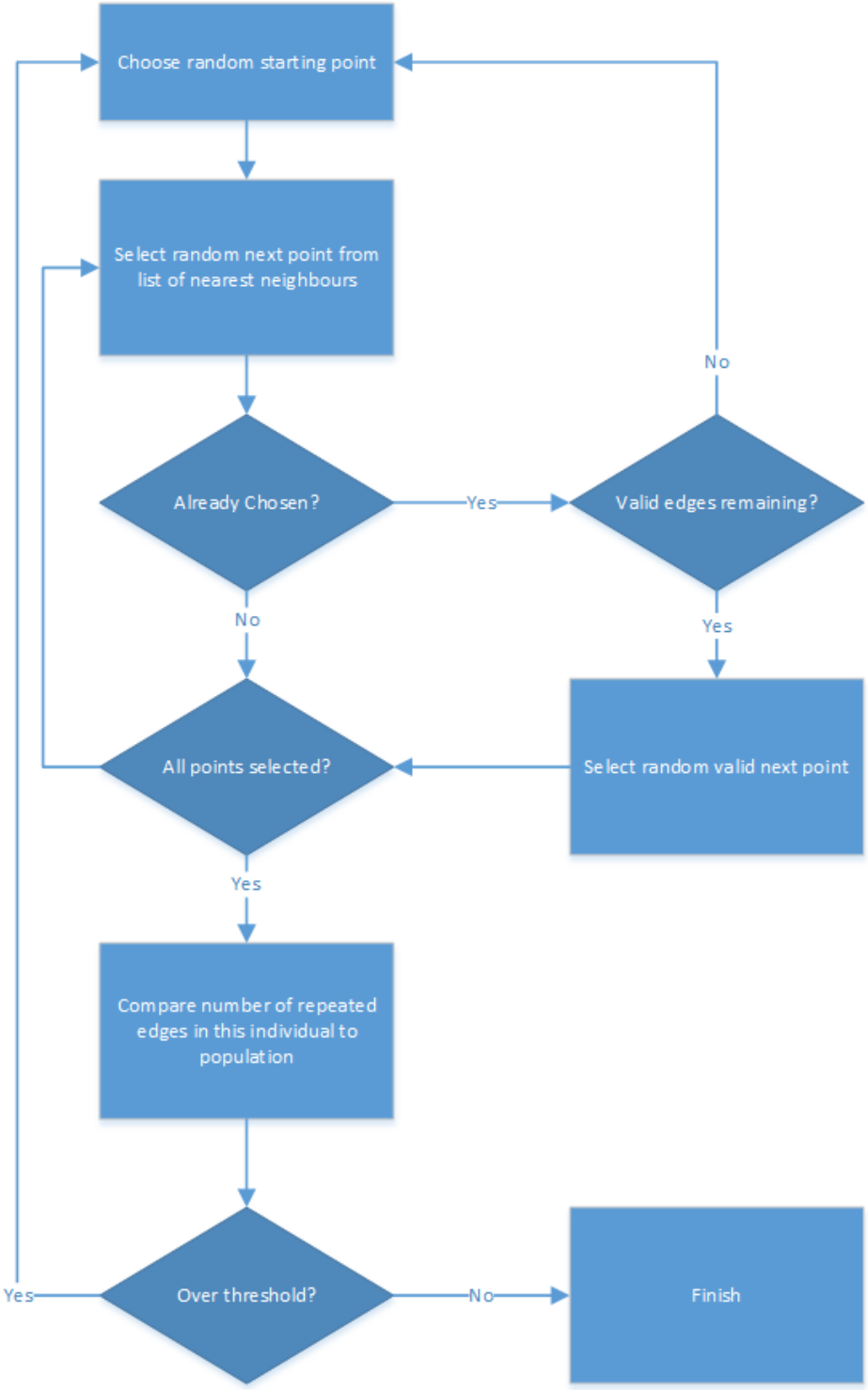


Figure 4-13: Initialisation process of an individual in the population

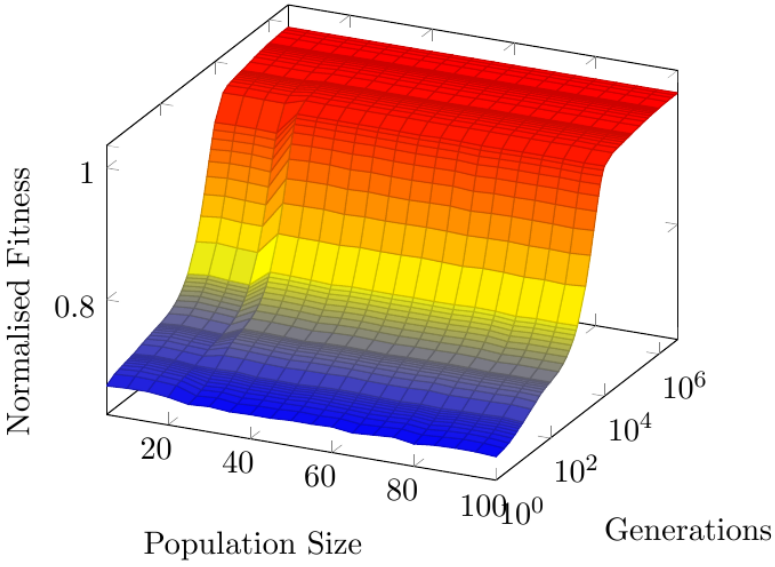


Figure 4-14: Effect of Population Size on Fitness for Up to 5 Million Generations.

of generations have been generated without an improvement. After testing the genetic algorithm it was decided to use 1000 generations without any improvement to the toolpath as the ending clause to the optimisation loop as this was shown to be sufficient to conclude no further improvements would ensue.

(i) Selection

The first step of the genetic algorithm after initialisation is to select two individuals to create a new individual from. Three types of selection operators were identified from the literature and tested in the genetic algorithm.

To test the performance of the three selection operators mentioned above, the genetic algorithm went through a Monte Carlo analysis with each of the three operators. A Monte Carlo analysis was performed to determine the number of improved individuals from two parent individuals that were chosen using the various selection operators. All the other operators of the genetic algorithm were kept constant and only the selection operator was changed. Each Monte Carlo analysis consisted of 1100 runs of the genetic algorithm. The error of the Monte Carlo analysis can be found by using the following formula:

$$E = \frac{z \times \sigma}{\sqrt{N}} \tag{4.11}$$

Where z is the confidence level, σ is the standard deviation and N is the number of runs in the Monte Carlo analysis. The confidence level that was used in the tests was 99% which corresponds to a z value of 3. The number of runs was 1100 and the highest standard deviation of the three sets.

The first strategy that was tested was a random selection of two individuals from the population without any selection criteria. This led to a very high percentage of new individuals being generated that did not provide an improved solution to the current highest ranking individual. From that conclusion it was decided to introduce some logic into the selection operator and use a rank selection operator. The population was ranked and ordered according to their fitness scores. The probability of each individual being selected was reduced by a constant the further the algorithm progressed down the ranked population. This method provided much better results at generating improved individuals. However there was still the possibility of there being a high amount of poor performing individuals in the population and the selection operator selecting these individuals. Therefore to reduce the chances of the selection operator selecting poor performing individuals, a fitness proportionate selection operator (also known as roulette wheel selection) was implemented. This method had the highest number of improved individuals being generated from the selected individuals. The results of the Monte Carlo tests can be seen in Figure 4-15. The fitness weighted selection operator outperformed the other selection operators and was therefore chosen as the final selection operator for the genetic algorithm.

The first step in the fitness proportionate selection operator is to sort the population from the highest ranking individual to the lowest. The total fitness of the population is then calculated. The probability of each individual being selected is then calculated using Formula 4.12.

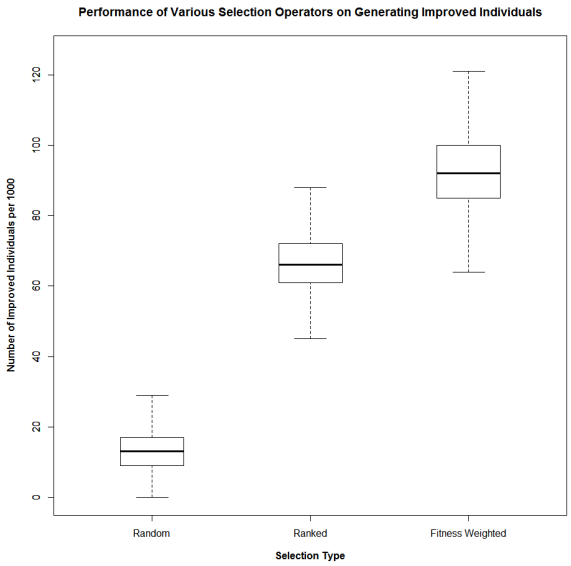


Figure 4-15: Box plot showing effectiveness of various selection operators

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{4.12}$$

Where f_i is the fitness of individual i in the population and N is the total number of individuals. The selection bounds for each individual will be cumulative weightings of the individuals ranked above it and the cumulative weightings including the current weighting. The algorithm then generates a random number between 0 and 1 to select one of the individuals in the population. It can therefore be seen that using this method, individuals with similar fitnesses will have similar probabilities of being selected.

Once the first individual is selected, it is removed from the selection pool and the selection algorithm is repeated to obtain the second individual. The two selected individuals are then analysed to assess the similarity between the two toolpaths in the same manner as was done in the population initialisation. If the two individuals are too similar, the secondly chosen individual is discarded and a new second individual is selected.

(ii) Crossover

To generate a new individual, the two parent individuals that were chosen in the selection stage will be combined. Unlike regular genetic algorithms, an order based genetic algorithm is very dependent on the order of the chromosomes in an individual. Therefore it is important to preserve as much of the genetic order of each parent as possible. It is the order of the genes that define the individuals path, therefore a higher percentage of new individuals will be generated with a lower fitness if too much of the order is lost. There are a large number of crossover operators that have been developed to efficiently mix the genetic information of the two parent individuals as can be seen in Chapter 3 Section 3.6.2.

The performance of any crossover operator is very dependent on the problem that the genetic algorithm is trying to solve. However it is impossible within the time frame of this reseach to test all the crossover operators and find the best performing operator for the toolpath generation problem. Therefore it was decided to find the top three performing crossover operators for genetic algorithms solving the travelling salesman problem. From the literature it was found that the three best performing crossover operators were the Edge Recombination Crossover(ERX), Partially Mapped Crossover(PMX), and the Order Crossover(OX).

The ERX operator focuses on preserving the edge information contained within the parent individuals. The edge information is analysed before crossover and used in the individual creation stage. The amount of crossover of each parent is randomly decided with each new gene. The ERX operator avoids generating illegal individuals by choosing one edge at a time and checking for illegal edges. The probability of a gene being used from one of the two parents is 50%. Therefore the new individual is created with half of the edges of each parent. This is a good method for equally crossing over two chromosomes and preserving the edges of both but can lead to issues with the overall order of the chromosome. This is because there is a high chance of alternating parent edges in the new individual. This is less of an issue if the two parent individuals are similar but is an issue if the two parent individuals are very different genetically. Because of this, the ERX crossover is more efficient towards the end of the genetic algorithm as it approaches convergence.

The PMX operator focuses on preserving the order within the parent chromosomes. This is done by copying one of the parent chromosomes and replacing a section of it with a length of the other parent's chromosome. This method can lead to illegal individuals and requires a second stage to remove any illegal edges in the chromosome. As the algorithm removes any illegal edges it also replaces a number of the edges that were present in the parent chromosomes. This process may result in a slight loss of parent edges but not as much as using the ERX operator.

The OX operator functions in a similar method to the PMX operator but differ in the way that the copied parent chromosome receives the donated genes from the other parent. The equivalent genes being donated are first removed from the copied chromosome and then shifted to remove these gaps to make room for the donor genes. This operator results in a slightly lower gene order retention when compared to the PMX operator.

To determine which of the three crossover operators will work best in the genetic algorithm for generating machining toolpaths, each of the three operators were tested in the genetic algorithm. Due to the modular nature of the genetic algorithm, it is possible to keep all of the other modules of the genetic algorithm constant and only vary the crossover operator. A Monte Carlo experiment was performed on each operator to identify the full effects on the genetic algorithm of each operator at various problem complexities and number of generations.

When generating a toolpath for complex feature shapes it is very difficult to identify whether the genetic algorithm has reached a global or local maximum. Therefore it would only be possible to determine the performance of the various operators relative to each other. A better method is to test the performance of the crossover operators using the Monte Carlo method on a square pocket of varying size. It is very easy to calculate the global optimal tool path for a uniform square grid. Using this feature it is possible to see how well the genetic algorithm performs with respect to the global optimum and not relative to one another. Another reason for using square pockets of varying sizes for the Monte Carlo experiments is to ensure that the only variable being altered is the number of points in the problem without any change in the complexity of the feature. The genetic algorithm might behave differently

between a perfectly square pocket and a slightly rectangular pocket.

The modified travelling salesman problem that the genetic algorithm is solving becomes factorially more difficult to solve with increased number of points. Therefore it is important to identify the maximum number of points to be used that will be solved in a timely manner and still represents a meaningful challenge for the genetic algorithm. To obtain a statistically significant result, the genetic algorithm will be run 10000 times for each Monte Carlo experiment.

The genetic algorithm was first tested with each crossover operator on a problem with the maximum number of points to determine how many generations the genetic algorithm should perform in the Monte Carlo experiment. Any simpler problem will only require an equal number of generations or fewer.

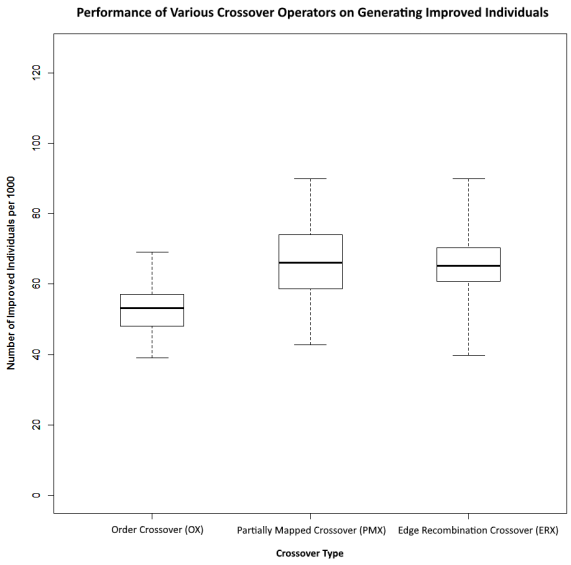


Figure 4-16: Box plot showing effectiveness of various crossover operators

The results of the Monte Carlo experiment for the three crossover operators can be seen in Figure 4-16. The ERX and PMX performed similarly in generating improved individuals when compared to the OX operator. However, the PMX operator is not as computationally intensive as the ERX operator. Therefore the PMX operator was chosen as the crossover operator to be used in the genetic algorithm due to it being much quicker to generate the

new individuals while still maintaining a similar performance to the ERX operator.

Now that the crossover operator has been selected for the genetic algorithm, it needs to be adapted slightly to solve the problem of toolpath generation. The PMX operator has been designed in such a way to retain genetic order and to avoid duplicate genes but does not take the feature of the part into account. Therefore it is still possible to generate paths that cross a feature boundary and would result in an illegal toolpath. To avoid this an extra check has to be performed for each new edge in the new individual to make sure that no subpath crosses a boundary line. This can be done fairly easily by comparing each edge to the distance matrix calculated at the initialisation of the genetic algorithm. This matrix also contains information on whether an edge is illegal or not. Once this has been done, it can be certain that the newly created individual will have a legal toolpath containing genetic information from both parent individuals.

(iii) Mutation

To ensure the genetic algorithm converges on an optimal solution, it is important to keep the available genes in the gene pool as diverse as possible. One way to do this is to insert new gene combinations into the gene pool by adding a mutation function into the genetic algorithm. As the algorithm is dealing with an ordered chromosome, mutating a gene value to a random number can cause a point to be visited twice and another point not at all. To solve this issue with ordered chromosomes, a special set of mutation operators were developed as can be seen in Section 3.6.2 of Chapter 3. To prevent illegal paths, the ordered mutation operators change the order of the genes randomly instead of the individual genes. This ensures that all points are still visited and no points are visited more than once.

From the literature it was found that the best performing mutation operators for an ordered genetic algorithm were the Exchange Mutation and Inversion Mutation operators. The exchange mutation operator works by randomly selecting a first gene and then selects a second gene in the set of points randomly. These two genes will then switch positions in the chromosome. This can result in four new edges being created or two new edges if the genes

were already located next to each other. The exchange mutation operator is very simple and preserves almost all of the order in the chromosome.

The inversion mutation operator uses a different method to alter the chromosome. Similar to the exchange mutation operator, two genes are chosen at random but instead of switching the two gene positions, all of the genes between the two selected genes are reversed in terms of their order. This method only creates two new edges in the chromosome while still retaining most of the order of the genes.

Both of these mutation operators work well in introducing new genetic edges into the chromosome, however in the problem set of generating toolpaths it is possible for both of them two produce illegal toolpaths whereby the toolpath crosses a feature boundary. This can be overcome by performing a check after mutation to assess whether any new edges cross any boundaries.

To avoid mutations of the chromosome resulting in an illegal path, a new method of mutating the current gene to one of the nearest neighbours of the previous gene was developed[163]. This method is a hybrid of the two previously discussed mutation operators with added logic to improve the effectiveness of the mutation operator. This method combines both the exchange and inversion mutation operators by switching two gene positions and inverting all the genes between the two chosen genes. The first gene is selected at random and the second gene is selected from a set of the nearest neighbours of the previous gene. The set of nearest neighbours includes the nearest N points to the gene selected for mutation and excludes any points that would result in an illegal edge. The nearest neighbour array is created in the initialisation of the genetic algorithm so that the calculation for the nearest neighbours to a given point is only performed once. This reduces the computational effort required by the mutation operator.

In the toolpath generation problem, if the objective function is path shortness, then the optimal subpath from one point to another will be two points with minimal distance between them. If the objective function is path straightness, then the optimal subsequent point to any given point will be a point in one of four directions to the original point. The

nearest neighbour (NN) method increases the probability of the mutation being successful by reducing the scope of the available genes that the selected gene can switch positions with. The following calculation verifies the benefit of using this method.

$$\text{Probability of Successful Mutation without NN Method} = \frac{1}{(XY^2 - XY)} \tag{4.13}$$

where X = points along X axis and Y = points along Y axis.

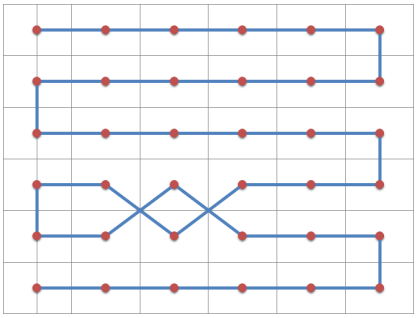
$$\text{Probability of Successful Mutation With NN Method} = \frac{1}{NXY} \tag{4.14}$$

where X = points along X axis, Y = points along Y axis and N = number of nearest neighbours. Therefore there is an $\frac{(XY-1)}{N}$ increase in probability of having a successful mutation when using the nearest neighbour method.

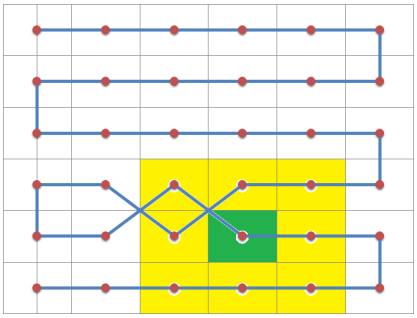
The sequence of figures from Figure 4-17a to Figure 4-17d illustrates the nearest neighbour mutation method. Figure 4-17a is an example of a toolpath which is not completely optimal. In Figure 4-17b the gene targeted for mutation is highlighted along with its nearest neighbours. The genetic algorithm will then select one of these nearest neighbours at random to become the next point in the toolpath order. In Figure 4-17c the optimal nearest neighbour is selected and its position is then switched with the original gene to ensure order is kept in the chromosome. The result of the mutation can be seen in Figure 4-17d where the two switched genes are highlighted. The toolpath is now of optimal length and direction.

The three mutation operators were tested using the same Monte Carlo method as was used in testing the various crossover operators. All of the other operators and variables were kept constant and only the mutation operator changed between tests. The tests were designed to verify which operator produced the most optimal results and also the efficiency of each operator by looking at how many generations were required to reach convergence in the genetic algorithm.

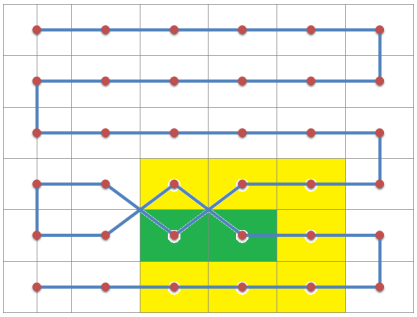
From the Monte Carlo experiment results in Figure 4-18 it was found that the toolpaths



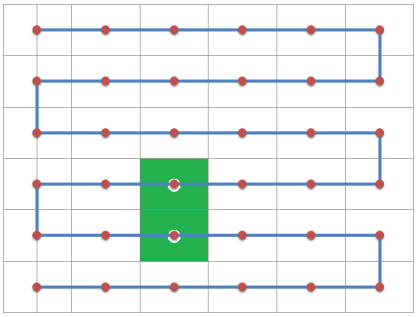
(a) Non-optimal Toolpath.



(b) First gene location with nearest neighbours.



(c) Nearest neighbour selected.



(d) Completed gene mutation.

Figure 4-17: Example mutation using the nearest neighbours operator.

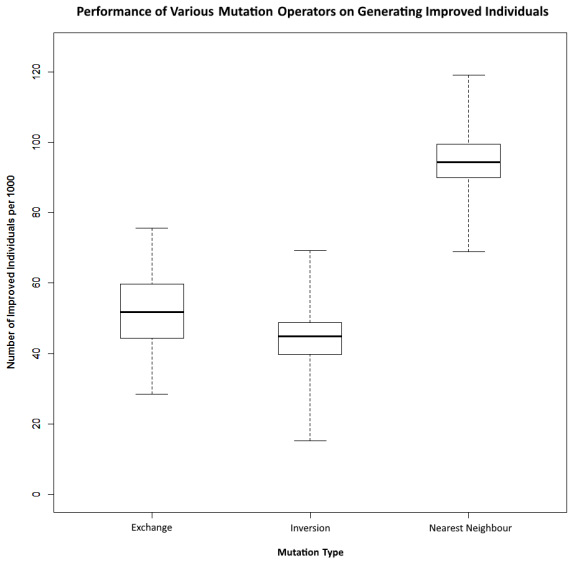


Figure 4-18: Box plot showing effectiveness of various mutation operators

adjusted by the nearest neighbours mutation operator were more effective at producing an improvement in the individual than the ones produced by the other mutation operators. Therefore it was decided to use the nearest neighbours mutation operator in the final design of the mutation module for the genetic algorithm.

(iv) Fitness Function

The fitness function is the most important stage of the genetic algorithm. It directs the flow of optimisation towards one or more specified goals. The objective goal can be set up as a minimisation function or a maximisation function. Secondary objectives can be included in the fitness function by adding a penalising or rewarding function to the fitness of an individual. It is important to keep the objective function simple as adding multiple variables and various rewarding/penalising functions results in a complex task of balancing the weighting of each objective function. As the fitness function module of the genetic algorithm is very complex and the problem of toolpath generation consists of various objective functions it was decided to discuss this topic in further detail in the following section.

4.4 Optimisation of the Toolpath

This section will discuss the various objective functions that were chosen and implemented into the fitness function for the genetic algorithm. There are a number of different qualities a CNC machining toolpath can possess. To demonstrate the capabilities of the genetic algorithm it was decided to investigate three different qualities of a machining toolpath and enable the genetic algorithm to easily switch between generating a toolpath with each of the three different qualities. The three selected qualities to be investigated for the research were the path length, straightness of the path and the average cutter engagement of the path. The mathematical models of these three objective functions were discussed in Chapter 4 Section 4.1. As some toolpaths require adherence to more than one quality, the effectiveness of a multi-objective function was also investigated.

4.4.1 Path Length

The main objective function in solving the problem of toolpath generation for machining is to minimise the toolpath length. If the feedrate of the machining tool is kept constant then by reducing the toolpath length, the time taken to machine a part will also be reduced. This results in an increased production rate and increased cost efficiency of manufacturing a part. This is under the assumption that all other machining variables are kept constant.

The total toolpath length is calculated by summing all of the individual subpath lengths of the edges contained within an individual. This is done by starting at the first gene of the chromosome and calculating the linear distance between the first gene and the subsequent gene. This process is iterated through the chromosome until all of the edge distances have been calculated and added to the total distance.

In a standard genetic algorithm this process would be repeated for each individual in each generation. This requires a lot of computational effort and in many cases a lot of the distance calculations are repeated many times. To overcome this and increase the efficiency of the algorithm, a distance array is used which contains all of the distance values for all of the possible edges for a given set of points. This requires the distance of each edge to be calculated only once. To further increase the efficiency of the algorithm, the genetic algorithm tracks changes in the chromosome during crossover and mutation and only updates the distances of edges that are altered. The fitness function now adheres to the path length objective function with the minimal amount of computational effort.

The drawback of performing the fitness function after the crossover module and mutation module is that the fitness function itself becomes less modular. This is not an issue if there is only one objective function, however if there is more than one objective function to choose from then it becomes more complicated to switch between the various objectives. To keep the fitness function modular with the more efficient computational method the crossover and mutation operators refer to the fitness function module with the information of altered edges to update the fitness of the individual.

Figure 4-19 illustrates how the fitness function is kept modular with the chromosome

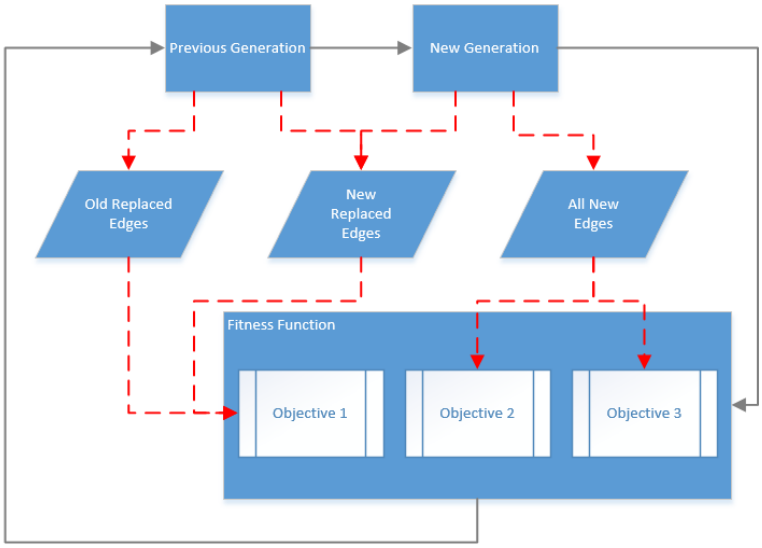


Figure 4-19: Modular fitness function with edge change tracking.

change tracking. The information of altered edges (red dotted lines) in the chromosome is fed into the fitness function for objective functions that only require this information. Any other objective functions will then be run after that if required.

The fitness function for the path length objective is calculated using Equation 4.15.

$$Fitness = \frac{1}{\sum_{i=1}^{n} X_{i,j} D_{i,j}} - \frac{1}{\sum_{i=1}^{n} X'_{i,j} D_{i,j}} \tag{4.15}$$

Where X is the set of new edges in the individual and X' is the set of old edges that were replaced through crossover and mutation. If the individual belongs to the first generation of individuals then the old set of edges will be empty and the new set will contain all of the edges in the individual. By using this fitness function, a toolpath can be generated which will find the shortest path through a set of points, as can be seen in Figure 4-20.

4.4.2 Path Straightness

The second toolpath characteristic to be considered to be used in an objective function is the overall straightness of the toolpath. As discussed in Chapter 4, sharp turns in a

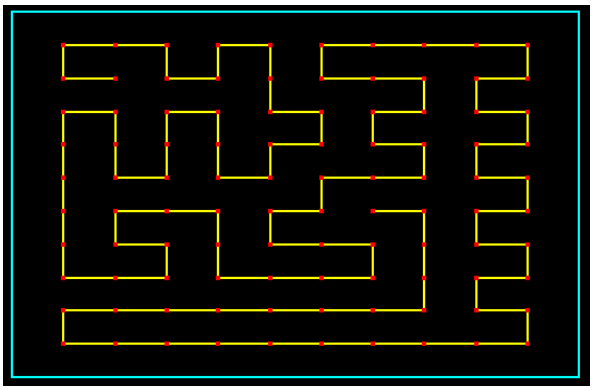


Figure 4-20: An example generated path with length optimisation.

toolpath can produce kinematic errors which can lead to errors in accuracy of the features of a machined part. Therefore it is important to reduce the number of sharp turns in a toolpath by developing an objective function which quantifies and minimises the sharpness of turns.

There are two ways in which the straightness of a path can be evaluated on. The first method is by counting the number of turns that occur throughout a toolpath. This is a discrete measurement as any change in direction of a subpath with respect to the previous subpath will count as a turn. The second method of measuring path straightness is to measure the change in angle of direction between each subsequent subpath. The total turn angle of the toolpath can then be obtained by summing the mod of all of the individual subpath angles.

Each method of calculating path straightness produces an optimal toolpath but each one follows a different machining strategy. Using the turn based approach to straightness calculation, a toolpath is generated which follows the bidirectional machining strategy. This is due to the fact that the objective function drives the toolpath to have long straight sections with sharp turns on the ends. The straight sections will always develop along the direction of the longest dimension of the feature as this will give the individual the best fitness. Using the turn based objective function the direction of the straight sections can change if necessary, for example the feature in Figure 4-21 requires a vertical direction of machining through the narrow area on the left and a horizontal direction for the rest of the feature.

An objective function which greatly favours consecutive subpaths in the same direction was developed to help drive the evolution of the toolpath towards the bidirectional strategy.

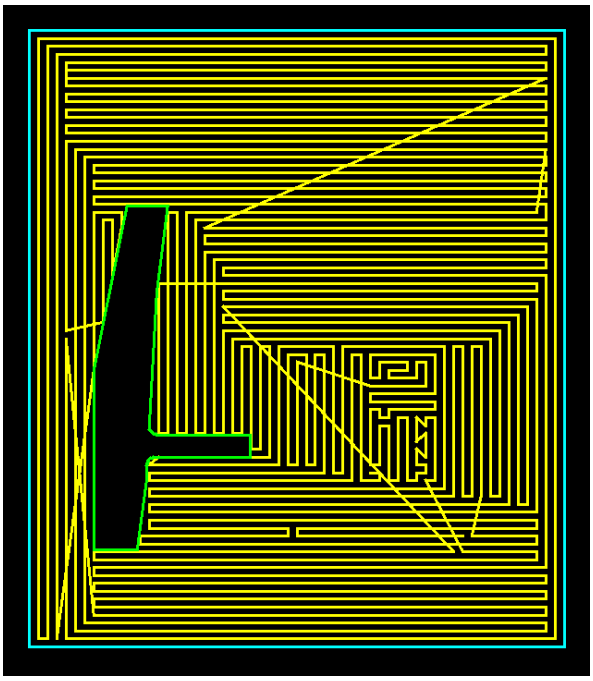


Figure 4-21: An example of how the direction of machining can change around a feature.

After testing a various methods of rewarding consecutive same direction subpaths it was found that rewarding each additional subpath exponentially outperformed the other methods. This is due to the fact that the objective function gives priority to extending the length of the longest subpaths already in the individual. Therefore extending a 5-point straight line to a 6-point line increases the fitness by 11 and reducing a 4-point straight line to a 3-point line reduces the fitness by 7 which results in an overall increase in fitness.

To implement this optimisation method into the fitness function, a series of logical steps were developed in the fitness function to produce the required toolpath with the bidirectional strategy. Figure 4-22 illustrates the logical steps the program undergoes to achieve this. The fitness function begins by iterating through the toolpath until two consecutive subpaths have the same direction. The program will then continue iterating through the points until a change of direction occurs and then will add the square of the number of consecutive points that had the same direction to the fitness. This process is continued until it has iterated through all of the toolpath's points.

The objective function can now generate a bidirectional toolpath for a given set of points.

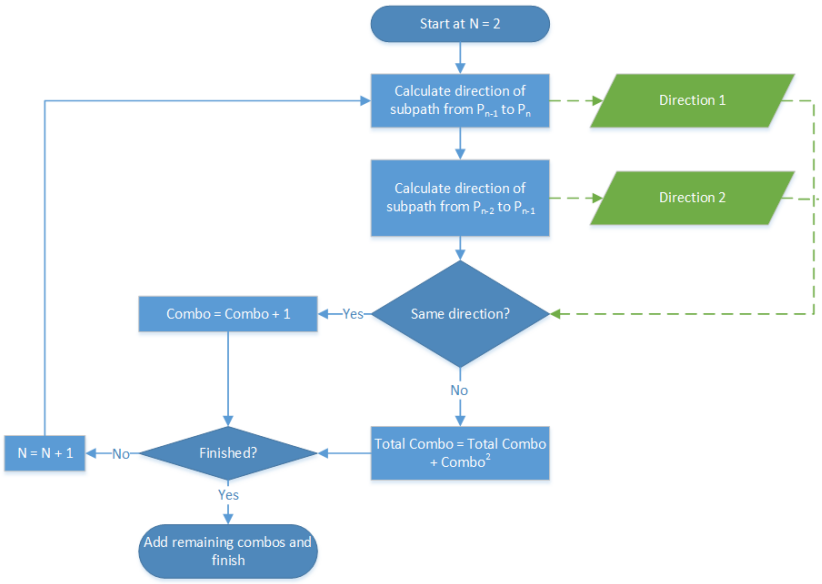


Figure 4-22: Turn based straightness optimisation part of the fitness function.

An example of a toolpath generated by using the turn based objective function for a square pocket can be seen in Figure 4-23.

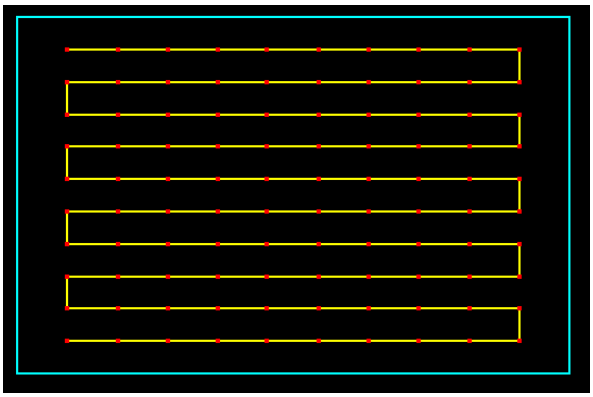


Figure 4-23: An example generated path with turn based straightness optimisation.

The angle method of calculating the path straightness results in a toolpath being generated that follows the spiral machining strategy. This is because the spiral strategy follows the contour of the feature towards the centre or vice versa and therefore naturally has the minimum amount of turning angle over the course of the toolpath. The objective function to generate the spiral type of toolpath mentioned will be a minimisation of the sum of the mod of the angular difference between consecutive subpaths along the toolpath. This objective

function can be seen in Equation 4.16.

$$\min \sum_{i=1}^{n} X_{i,j}|\theta| \tag{4.16}$$

Figure 4-24 is an example of two consecutive subpath with θ_1 and θ_2 representing the individual angles of each subpath. The position of the paths can be rearranged as shown in Figure 4-24 to visually represent the total angle change between two subpaths. The angular difference can then be calculated by subtracting θ_1 from θ_2. The atan2 programming function was used to calculate each value of θ. The atan2 function was chosen as opposed to calculating the dot product between the two paths as it has a higher accuracy when performing floating point arithmetic. The atan2 function also avoids increasing errors as two subpaths approach being parallel or perpendicular to each other which occurs very frequently in machining toolpaths. The atan2 function is defined in Equation 4.17.

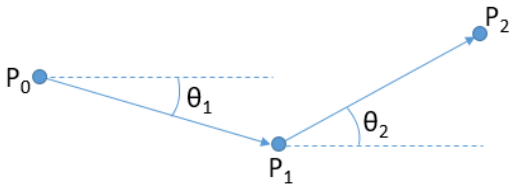


Figure 4-24: Angle of each subpath.

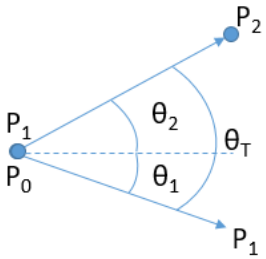


Figure 4-25: Angle difference between two subpaths.

$$\text{atan2}(y,x) = \begin{cases} arctan(\frac{y}{x}) & \text{if } x > 0, \\ arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{4.17}$$

As seen in Equation 4.17, the result is undefined when x = 0 and y = 0, but this is avoided in the TSP problem there is no legal subpath that would begin and end in the same position. The range of results produced by the atan2 function is between $-\pi$ and π. The total angle change between the two subpaths (denoted by θ_T) can now be calculated by inputting the x and y coordinates into the atan2 function as described in Equation 4.18.

$$\begin{aligned} \theta_T &= \text{atan2}(A,B) - \ \text{atan2}(C,D) \qquad (4.18) \\ \text{where A} &= y_n - y_{n-1} \\ \text{B} &= x_n - x_{n-1} \\ \text{C} &= y_{n-1} - y_{n-2} \\ \text{D} &= x_{n-1} - x_{n-2} \end{aligned}$$

The result from Equation 4.18 will be between -2π and 2π. The angle given by Equation 4.18 can either correspond to the major or minor arc subtended between the two subpaths, however only the minor arc angle is required for the objective function. Therefore Equation 4.18 needs to be adjusted to only produce results in the $-\pi$ to π ratio. This adjustment can be seen in Equation 4.19 which only calculates the angle of the minor arc for θ_T.

$$\theta_T = \begin{cases} \theta_2 - \theta_1 - 2\pi & \text{if } \theta_2 - \theta_1 > \pi \\ \theta_2 - \theta_1 + 2\pi & \text{if } \theta_2 - \theta_1 < -\pi \\ \theta_2 - \theta_1 & \text{if } \pi \leq \theta_2 - \theta_1 \leq \pi \end{cases} \tag{4.19}$$

The objective function minimises the sum of all the subpath angles, therefore it is important to obtain the mod of the individual subpath angles. This is because the objective function relies on the magnitude of the angles instead of the direction.

Now that the angles can be properly identified, the objective function iterates through all of the grid points whilst summing all of the angles calculated using Equation 4.19. The fitness of the species is then defined by Equation 4.20.

$$Fitness = \frac{1}{\sum_{i=1}^{n} X_{i,j}|\theta_T|} \tag{4.20}$$

By minimising the objective function, a spiral toolpath can now be generated for a set of grid points. An example toolpath generated by using the angular straightness objective function for a square pocket can be seen in Figure 4-26.

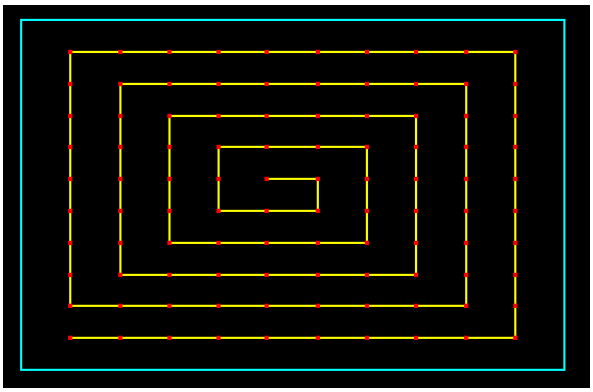


Figure 4-26: An example toolpath generated using the angular objective function.

4.4.3 Tool Engagement

The final toolpath characteristic considered to be used as an objective function is the engagement of the tool throughout the toolpath. The tool engagement influences the formation of chips and thus affects the surface quality of the part. A high value of tool engagement throughout the toolpath will result in a higher value of material removal which will reduce the overall machining time. However the higher the tool engagement the quicker the tool will wear.

Due to the advantages and disadvantages of having a high or low tool engagement, the tool engagement will be set at a target value that should be kept consistent throughout the toolpath. This target value will be a balance between the various effects the tool engagement produce. Therefore the objective function will not be an overall minimisation/maximisation but a minimisation of the deviation from this target value.

To calculate the consistency of the tool engagement over the full toolpath, the amount of material engaged the tool at any point throughout the toolpath needs to be calculated first. To do this the area of the tool and material need to be discretised. Ordinarily the tool engagement calculation considers the area of material engaged with the tool as seen in Figure 4-27, however as the cut depth is kept constant throughout each layer of machining the calculation can be simplified to just the amount of the tool's circumference engaged with the material.

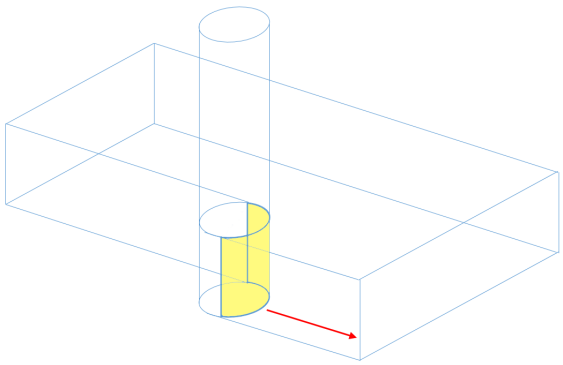


Figure 4-27: Area of tool engaged with the material.

As the tool engagement calculation has been reduced from a 3D to a 2D environment, the

discretisation of the tool and material can be reduced to a two dimensional problem. The tool engagement can be assesed by converting the layer of material into a grid of pixels where each pixel is defined as either containing material or not. The tool is also discretised and then rastered over the pixels to identify which pixels of material are in contact with the tool throughout the toolpath.

The accuracy of the tool engagement calculation is dependent on the resolution of the grid and the size of the tool. The accuracy of the calculated tool engagement increases as the resolution of the grid increases. However increasing the resolution of the grid also increases the computational effort required to perform the tool engagement calculations. The accuracy of the tool engagement value can be evaluated by determining the number of pixels used to model the circumference of the tool and then finding the ratio of one pixel to the full circumference. Since the distance between any two neighbouring pixels can only be 1 or $\sqrt{2}$, the number of pixels required to represent a circle with radius r is between $\sqrt{2}\pi r \leq x \leq 2\pi r$. The accuracy of the tool engagement calculation is reduced with fewer points. Therefore by using the lower bounds of pixels required to draw a pixelated circle the accuracy can be calculated using Equation 4.21.

$$\text{Accuracy} = \frac{Resolution}{\sqrt{2}\pi r} \tag{4.21}$$

If the radius of the tool and accuracy required are known, then Equation 4.21 can be rearranged to give the minimum resolution of the grid necessary to achieve the required accuracy.

The pixel grid is generated using the same method to generate the grid of points for the feature with a few changes. The number of pixels will be dependent on the resolution which is defined by the required accuracy of the calculation. Also instead of including or excluding a point, the pixel will be 0 if it is outside of the feature and a 1 if it is inside of a feature. The pixel information can be stored using two dimensional bit array.

The tool circumference is discretised into the same two dimensional environment by using the Bresenham circle algorithm[164]. The Bresenham circle algorithm identifies all of the

pixels on the circumference of a circle with a given radius and centre point as seen in Figure 4-28. Therefore the tool engagement at any point of the path can be determined by the ratio of pixels of the tool's circumference overlapping material to the total pixels on the tool's circumference.

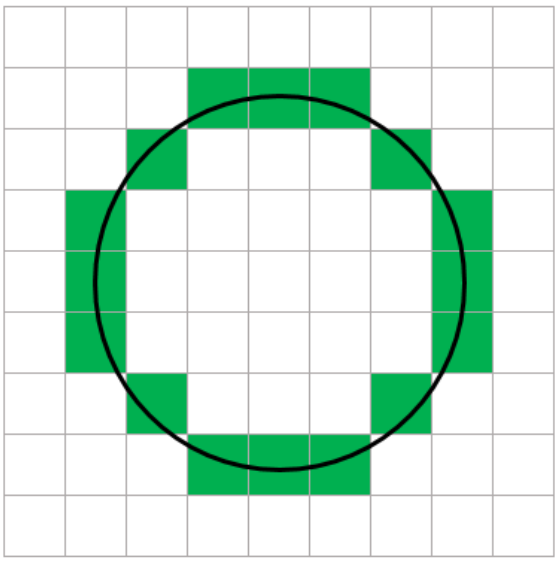


Figure 4-28: An example of a discretised circle using Bresenham's circle algorithm.

Now that the tool engagement can be calculated for a single point in the toolpath, the same method has to be repeated along the entire toolpath to determine the consistency of the tool engagement. To do this, the average tool engagement along each edge of the toolpath has to be calculated. The subpath can be discretised into the pixel grid by using Bresenham's line algorithm[165]. Bresenham's line algorithm identifies which pixels are included in a linear path between two points as illustrated by Figure 4-29

The tool circumference can now be rastered along the discretised path and the tool engagement calculation can be performed for each iteration of the path. The average tool engagement between two points can be obtained by summing all of the tool engagement values and dividing by the number of iterations in the path. The fitness function of the genetic algorithm can now iterate through all of the points in the grid and calculate the average tool engagement for each subpath between two points and adjust the fitness of the individual

according to the calculated values. The individual values of tool engagement is stored as well as the average for each subpath to be used by the fitness function as required.

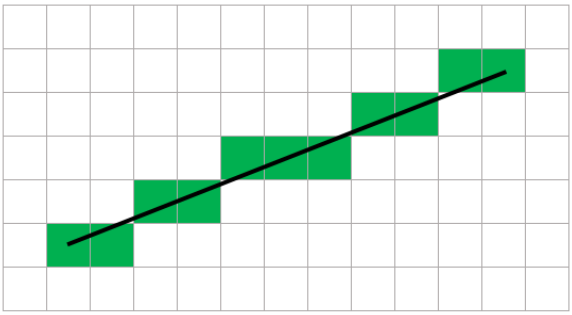


Figure 4-29: An example of a discretised path using Bresenham's line algorithm.

For this objective function, the fitness of an individual is penalised or rewarded depending on the value of tool engagement for each subpath. The individual is rewarded for having a smaller deviation from the target tool engagement but penalised if the tool engagement is greater than the target tool engagement. The fitness is penalised for tool engagement greater than the target as high tool engagements can cause higher tool wear and damage. Although higher tool engagement increases the material removal rate and hence increases productivity, it can be assumed that if the tool engagement target is lower than the maximum then the productivity is not the main priority.

To avoid tool engagement higher than the target, the penalty applied to the fitness will increase exponentially the greater the tool engagement is from the target. The reward applied to the fitness for having a tool engagement near the target will be the inverse of the difference between the actual tool engagement and the target tool engagement. Therefore the objective function can be defined as seen in Equation 4.22

$$\text{Fitness} = \sum_{1}^{n} X_{i,j}(\text{A} \ - \text{B}^2) \tag{4.22}$$

$$\text{where A} \ = |\text{TE}_c - \text{TE}_t|$$

$$\text{B} \ = \begin{cases} \text{TE}_c - \text{TE}_t & \text{if TE}_c > \text{TE}_t \\ 0 & \text{if TE}_c < \text{TE}_t \end{cases}$$

By minimising the objective function in Equation 4.22, a toolpath can be generated for a grid of points with the tool engagement being as close to the target as possible without going over the target. An example toolpath generated using the tool engagement objective function for a square pocket can be seen in Figure 4-30

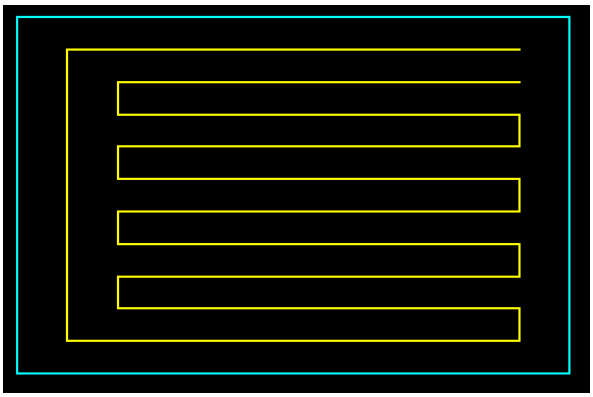


Figure 4-30: An example of a generated toolpath using the tool engagement objective function.

As seen in Figure 4-30, the toolpath is only optimised for tool engagement and not length or straightness. However Figure 4-31 illustrates the distribution of the tool engagement for all of the subpaths seen in Figure 4-30

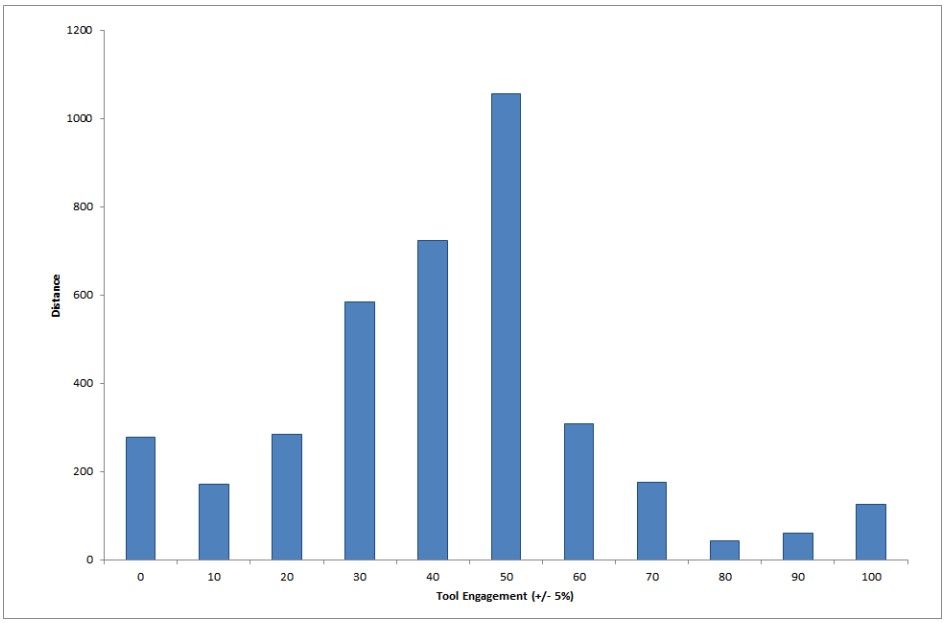


Figure 4-31: The distribution of tool engagement over the toolpath shown in Figure 4-30.

Chapter 5

Design of Test Cases for Validation

5.1 Introduction

This chapter will discuss the design of the test cases that will be used to validate the models developed in this research as well as the performance of the genetic algorithm. The quality of the solutions generated by the genetic algorithm will alongside the flexibility of the genetic algorithm will be the taken into consideration when designing the test cases.

5.2 Developing the Test Parts

5.2.1 Test Part Inspired by Aerospace Component

To test the toolpath generation capabilities of the genetic algorithm, a test part needs to be designed that contains a variety of features and machining operations which would commonly be found in a manufacturing setting. An industrially inspired aerospace test part was used by STEP Tools known as the "fishhead"[166]. The fishhead part is an aerospace component produced by Airbus and was designed to be machined using a 5-axis machine tool. A CAD model of the fishhead part can be seen in Figure 5-1.

As the fishhead part contains features that can only be machined using a 5-axis machine

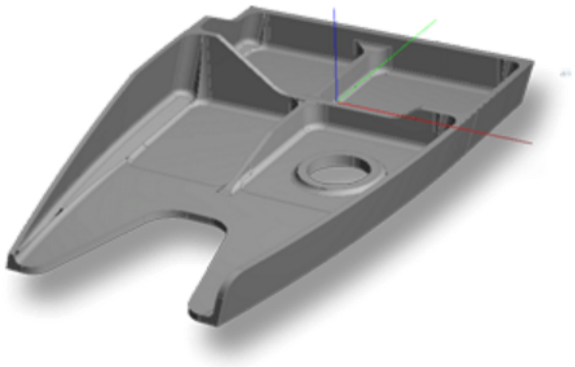


Figure 5-1: The original fishhead part.

tool, the fishhead design needs to be adapted so that features it contains are prismatic. The simplest way to convert the fishhead part into a 2.5D design is to remove all of the sloped feature boundaries and ensure that all the feature boundaries are perpendicular to the X-Y plane. The adapted fishhead part can be seen in Figure 5-2 with the dimensions for the part illustrated in Figure 5-3.

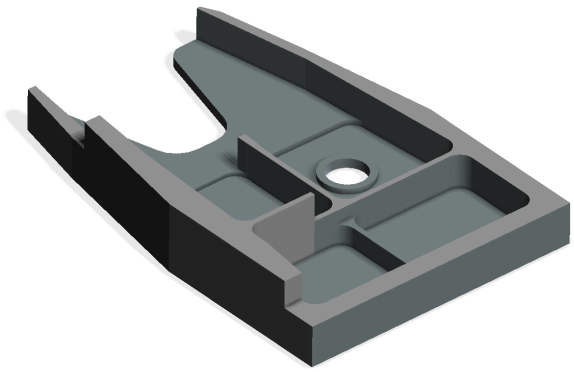


Figure 5-2: The adapted fishhead test part.

The fishhead part contains a variety of features defined by the STEP-NC standard. There are several closed pockets, open pockets, bosses and a hole. The fishhead part requires ten machining operations to finish the part including facing, pocket machining, contouring and boring. Three different tools are also required to machine the various features of the part. The feature types, required machining operations and tools used in this test case are all commonly found in manufactured parts. Therefore it can be said that the fishhead part is a

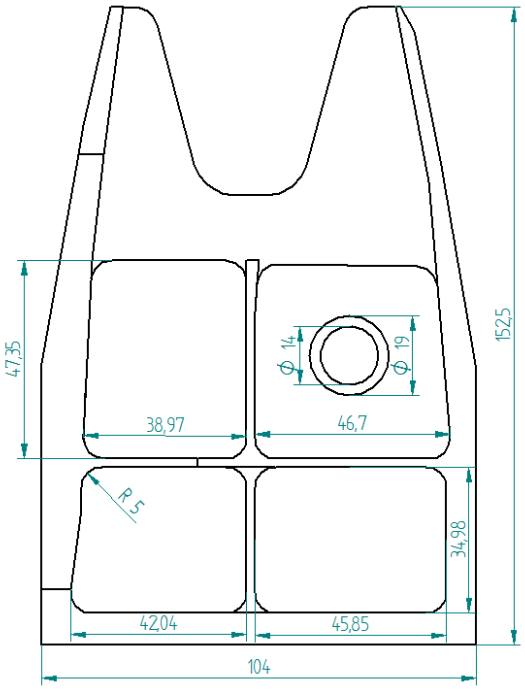


Figure 5-3: The dimensions of the adapted fishhead test part.

satisfactory choice of design to test the flexibility and performance of the genetic algorithm at generating machining toolpaths.

5.2.2 Test Part with Sloped Feature Boundary

A second test part was also designed to test the genetic algorithm's capability of generating machining toolpaths for non-prismatic features. As seen in Section 3, the boundary of a feature defined in the STEP-NC format allows for a slope to be specified for the feature boundary. Therefore one of the features contained within the fishhead part was modified to contain a sloped boundary. The final test part can be seen in Figure 5-4.

The scallop height is set at 0.5mm which will be used by the genetic algorithm to identify the correct cut depth for each layer. The actual scallop height generated by the genetic algorithm will be compared to the target scallop height. This will validate the method for generating toolpaths for features with a sloped boundary.

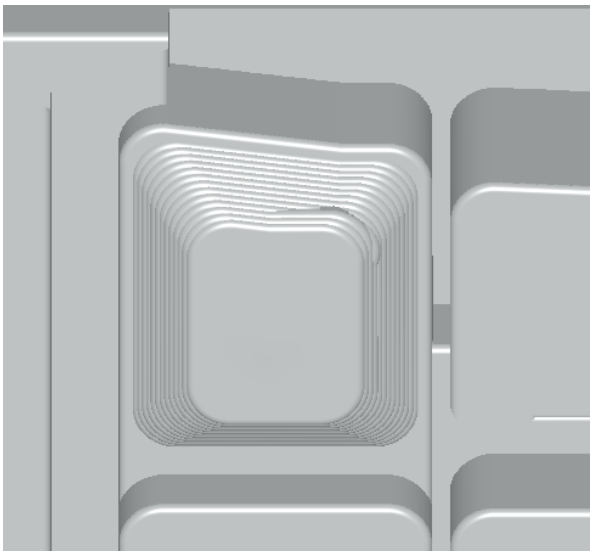


Figure 5-4: The test part containing a feature with a sloped boundary.

5.3 Genetic Algorithm Performance

A set of tests need to be designed to validate the objective functions developed for the genetic algorithm and analyse their performance. As discussed in Section 3.10, the quality of the solutions generated by the genetic algorithm need to be compared to currently used methods of generated toolpaths for the same part as well as the globally optimal solution with respect to each objective function.

5.3.1 Comparison to Established Methods

To compare the generated toolpaths from the genetic algorithm to currently used methods, three different commonly used CAM software packages were identified and selected. The CAM software packages that were selected are: NX developed by Seimens, HSMExpress developed by Autodesk, and iMachining developed by SolidCam. The performance of the genetic algorithm developed in this research will also be compared to the genetic algorithm developed by Car in 2006 which was designed to generate a machining toolpath for rectangular grids of varying sizes.

The same model of the fishhead test part in Figure 5-2 will be used for each CAM soft-

ware package. To compare the CAM generated toolpath to the genetic algorithm generated toolpath, two types of machining strategies will be used: bidirectional(zig-zag) and spiral. The machining parameters will be kept constant when generating the toolpath using the various software packages. Two tools will be used to machine the part, a 40mm facemill and a 10mm endmill. The cut depth is set to 2mm and the tool engagement set at 50%. Only the roughing toolpaths will be generated using the CAM software packages. The generated toolpaths will be compared on their length for both the bidirectional and spiral cases whereas the number of turns and total turn angle will be compared for the bidirectional and spiral cases respectively.

The study performed by Car attempted to generate toolpaths in a similar method to the one used in this research. Therefore the performance of the genetic algorithm developed in this study can be compared to the one developed by Car. Car described four test cases used in the study to test the genetic algorithm that was developed to generate toolpaths. The test cases consisted of various rectangular uniform grids with dimensions as follows:

- 10 x 10 cells
- 100 x 100 cells
- 100 x 75 cells
- 100 x 80 cells

The results published in the study contained the average time taken to obtain the generated toolpath as well as the fitness of the toolpath as a percentage of the global optimal solution. Although this study was performed in 2006 and the hardware used to obtain the results are outdated compared to modern technology, the processor clock rate is actually higher than the one used in this research (3GHz vs 2.6GHz). Therefore the only meaningful comparison between the genetic algorithm in this research and the genetic algorithm developed by Car will be between the fitness values of the generated solutions. However, the time taken to generate the solutions for both algorithms will still be included in the results in the following chapter.

5.3.2 Validating Genetic Algorithm Objective Functions

To determine the performance of the objective functions in the genetic algorithm, a set of tests need to be designed to compare the generated results to the global optimum for each individual objective. Three test cases were chosen from the TSPLIB database to test the performance of the path length objective function. The QA194, XIT1083, and FI10639 TSP problems as defined by the TSPLIB were chosen as the test cases. The three chosen TSP problems can be seen in Figures 5-5, 5-6, 5-7.

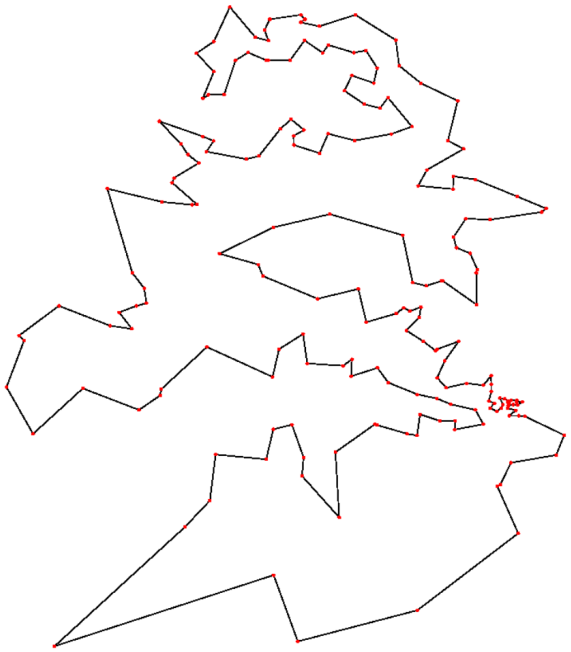


Figure 5-5: The QA194 TSP problem with the optimal tour.

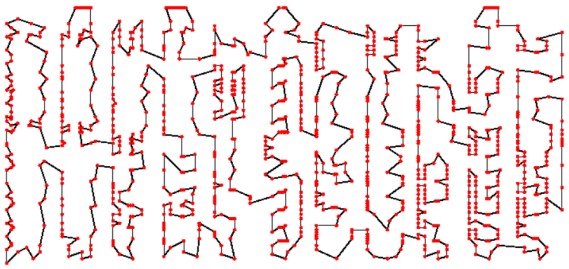


Figure 5-6: The XIT1083 TSP problem with the optimal tour.

These three TSP test cases were chosen as they all contained sets of points with an even

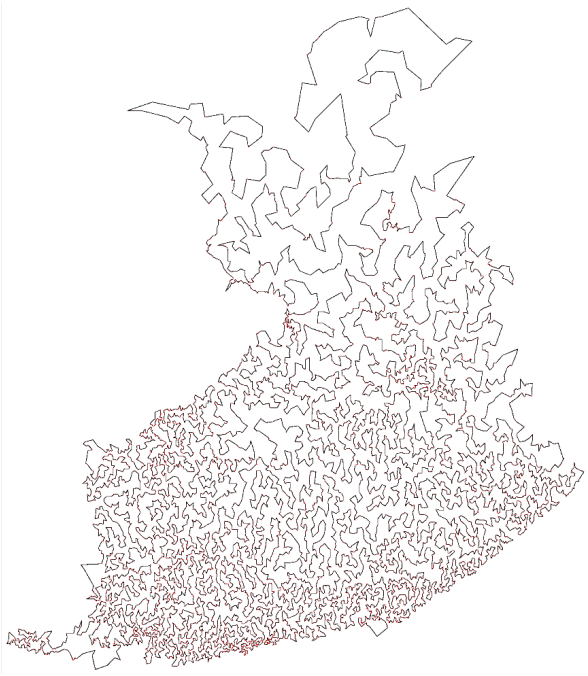


Figure 5-7: The FI10639 TSP problem with the optimal tour.

distribution. The number of points contained in each TSP was also a factor that was taken consideration. The three test cases are used to represent a trivial, intermediate and difficult problem for the genetic algorithm to solve. As a frame of reference, the largest set of points for a feature on the fishhead test part contained 1096 points. The QA194 TSP contains 194 points which is a small problem in comparison. The XIT1083 TSP contains 1,083 points which is very similar to the largest problem set for the fishhead part. The FI10630 TSP contains 10,630 points which is roughly ten times the size of the previous TSP. The size of this problem is far larger than would be expected to handle when looking at common ratios for tool sizes to feature sizes.

The genetic algorithm has been developed to optimise paths for uniform grids, however the TSPLIB does not contain any TSPs with a uniform grid. One slight adjustment has to be made to the genetic algorithm so that the number of neighbours that are considered includes the entire set of points for the problem instead of the 24 points that is normally considered. This is due to the fact that a cluster of more than 24 points can be present in the TSPLIB problems which results in some of the points in the cluster not being considered for mutation.

This can reduce the quality of the solutions as it leads to local optima. Altering the number of neighbours to be considered for the mutation algorithm does not fundamentally change the behaviour of the genetic algorithm but does increase the time required to converge on an optimal solution.

The average path length produced by the genetic algorithm over all of the Monte Carlo trails will be compared to the optimal path length as stated in the TSPLIB. This will be the only metric compared between the two as the standard travelling salesman problem only defines the length as its main optimisation objective.

The path straightness and tool engagement objective functions are unique to machining toolpaths. Unfortunately there is no database of standard TSP problems where the global optimum has been identified for the straightness and tool engagement objective functions. Therefore a set of test cases need to be designed in order to validate these two objective functions.

A set of uniform grids of increasing size were chosen to test the performance of the path straightness and tool engagement objective functions. The optimal toolpaths for these uniform grids will be identified by using a brute force search algorithm. These results will then be compared to those generated by the genetic algorithm.

This set of tests was also designed to identify the effect that increasing the number of grid points in the set has on the quality of solution generated by the genetic algorithm. This was done by generating square grids with NxN points where N ranged from 5 to 50 points. This produced problems ranging from 25 to 2500 points. The number of generations for the genetic algorithm to run for was defined by the number of generations required to converge on an optimal solution for the largest problem. It was found that 5 million generations was required to converge on an optimal solution for a square grid of 2500 points. Therefore the test for each problem will run for 5 million generations. The fitness of the population will also be sampled for the generations in the set of values defined by Equation 5.1

$$n \times 10^{i} \text{ where } \{(n, i) \colon n \in \mathbb{N}_{<10}, i \in \mathbb{N}_{<7}\} \tag{5.1}$$

As genetic algorithms are a stochastic optimisation method, it is important to assess the statistical variance in all of the results generated by the genetic algorithm in this research. Therefore in the same manner as discussed in Section 4.3.3, the Monte Carlo method will be applied when generating the results from the various tests described in this chapter. This will ensure that the results presented in Chapter 6 are an accurate portrayal of the genetic algorithms capabilities.

5.4 Summary

This chapter discussed the process of designing two test parts to be used in validating the models developed in this research. The first test part is a prismatic adaptation of an aerospace component which will be used to validate the genetic algorithms capability of handling a variety of geometries and machining processes.

Chapter 6

Machining Results with Comparison to Established Algorithms

6.1 Introduction

This chapter will provide the results obtained from the validation and testing procedures outlined in Chapter 5. The experimental data of each test case will be analysed in detail. Section 6.2 analyses the performance of the developed genetic algorithm, Section 6.3 covers the validation of the objective functions and Section 6.4 evaluates the performance of the computational platform to current methods.

6.2 Genetic Algorithm Performance

The behaviour and functionality of the genetic algorithm was tested.

6.2.1 Effects of Number of Grid Points

To analyse the effect the size of the problem has on the solution quality of the generated toolpaths, a set of tests were performed on the genetic algorithm where the number of grid points in the problem were increased with each test. The results of these tests can be seen in Figure 6-1.

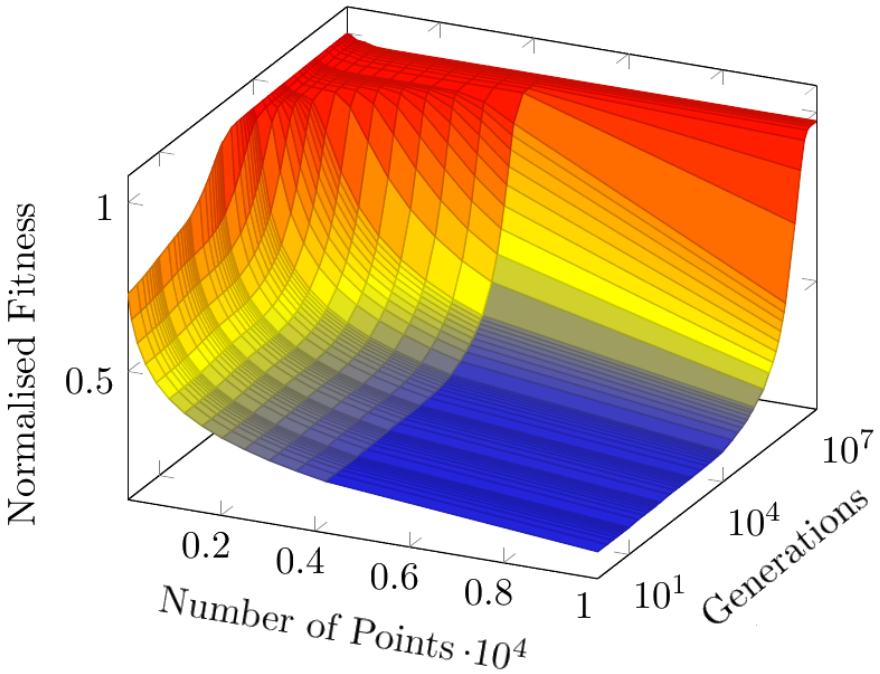


Figure 6-1: Fitness over all generations for point sets up to 10000 points.

It can be seen from Figure 6-1 that the fitness of the initial population decreases as the number of grid points increases. This is due to the increasing complexity of the problem as more grid points are added. The algorithm to generate the initial solutions is a basic nearest neighbour algorithm which performs better with fewer points in the problem.

The fitness of the generated paths after 10 million generations for point sets up to 10,000 points can be seen in Figure 6-2.

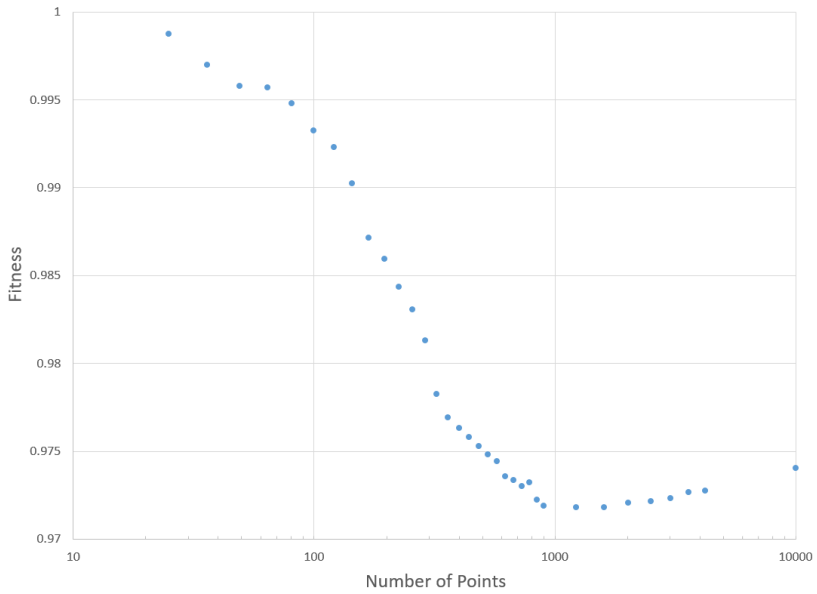


Figure 6-2: Fitness of generated paths for various point set sizes after 10 million generations.

It can also be seen that the number of generations required to reach convergence in the genetic algorithm increases as the number of grid points increases. This is due to the core part of the genetic algorithm staying constant despite the complexity of the problem. The mutation operator switches a set of edges or reverses them, therefore increasing the number of edges in the problem will require more generations to switch the required number of edges to reach convergence. This will naturally also require more time to generate an optimised toolpath for larger problems.

This is further illustrated in Figure 6-3 where the time taken to iterate through 1000 generations was tested for sets of grid points between 25 and 900 points in each set. The orange, blue and grey lines are for the maximum, average and minimum times respectively. The test was run 2000 times for each set of grid points. It can be seen that the time taken to iterate through 1000 generations increases exponentially with the number of points.

6.2.2 Variation in Generated Solutions

A set of tests were performed to analyse the variation in the solutions generated by the genetic algorithm due to it being a stochastic process. A toolpath was generated for a 100x100 grid of

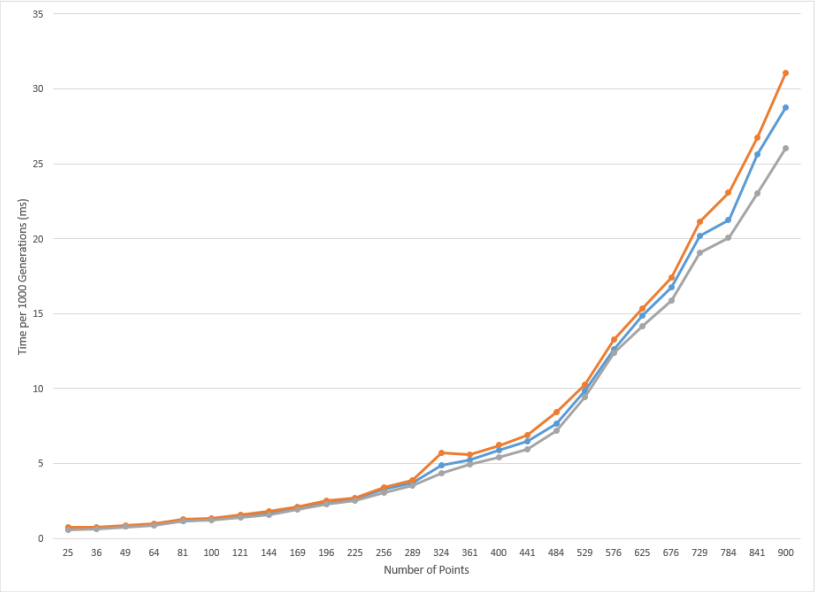


Figure 6-3: Time taken to iterate 1000 generations for increasing number of points.

points and repeated 2000 times to identify the spread of fitness values for the final solutions generated by the genetic algorithm. Figure 6-4 uses a histogram to illustrate this spread over the runs performed.

The fitness of the generated solutions varied by no more than 0.58% from the average solution quality. The spread of the fitness values followed a normal distribution with the average solution quality having a 97.3% fitness value.

Figure 6-5 shows the spread of fitness values over the generations of a run with the average, maximum and minimum values of fitness for each generation after 2000 runs. It can be seen that the spread is reduced as the genetic algorithm converges on a solution. This is due to the genetic algorithm quickly reducing the large solution space and focusing on the local/global minimums of the solution space.

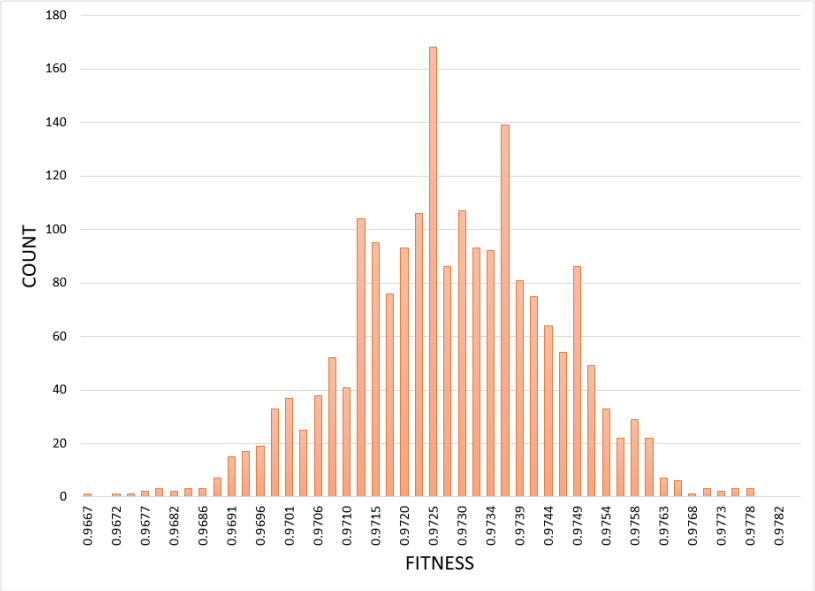


Figure 6-4: The distribution of quality for the generated solutions.

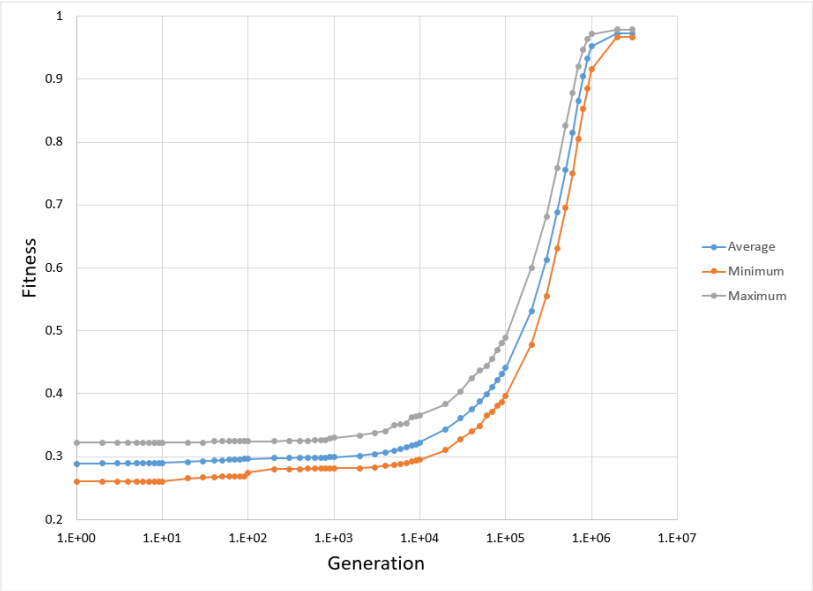


Figure 6-5: The spread of fitness over 3 million generations.

6.2.3 Scallop Height in Non-Prismatic Features

The toolpath generated by the genetic algorithm for the test case which contained a feature with a sloped boundary can be seen in Figure 6-6.

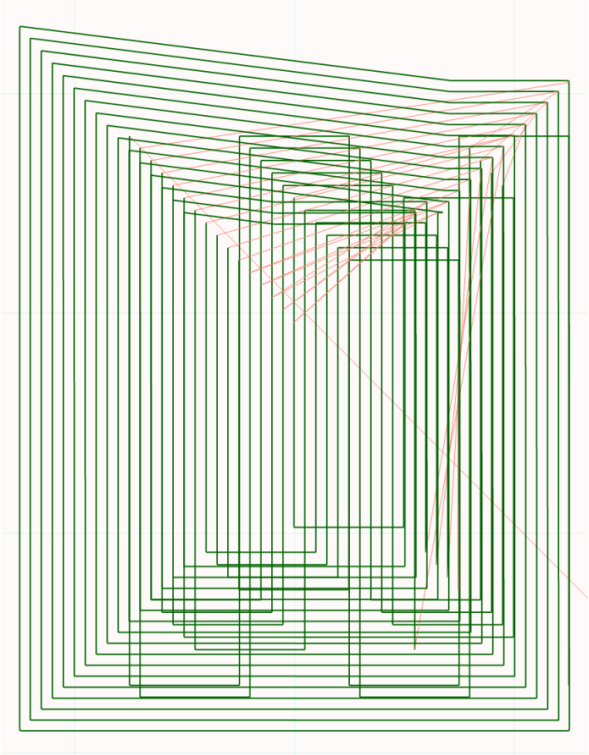


Figure 6-6: Toolpath generated by GA for 3D pocket.

The generated toolpath was then simulated using CNC Simulator to assess the end result of machining the workpiece with the generated toolpath. The results of the simulation can be seen in Figure 6-7.

The computational platform was able to generate the required points for the feature at each cut depth with the adjusted boundary at each layer. The scallop height produced by the generated toolpath was equivalent to the scallop height specified as the input parameter into the system as can be seen in Table 6.1.

Table 6.1: Comparison of scallop height between specified and actual values.

	Specified	Actual
Slope Height	0.50mm	0.48mm

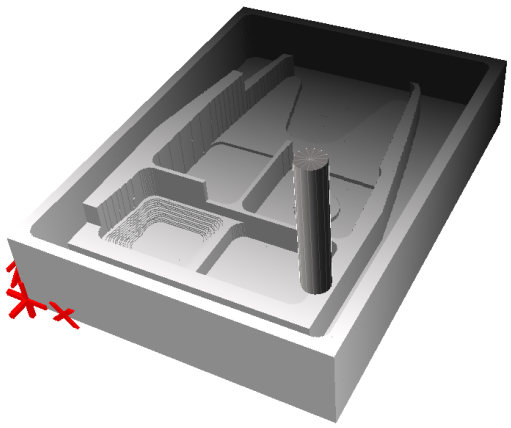


Figure 6-7: 3D Model of machined pocket with sloped boundary.

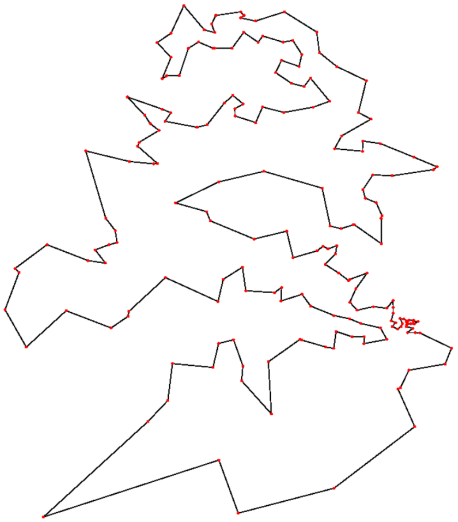
6.3 Validation of Objective Function Models

This section will present the results obtained from the test cases designed to verify the behaviour of the objective functions. The three objective functions tested were the path length, path straightness and tool engagement. A set of tests were also performed to assess the performance of the multi-objective function where all three objective functions previously mentioned were used simultaneously.

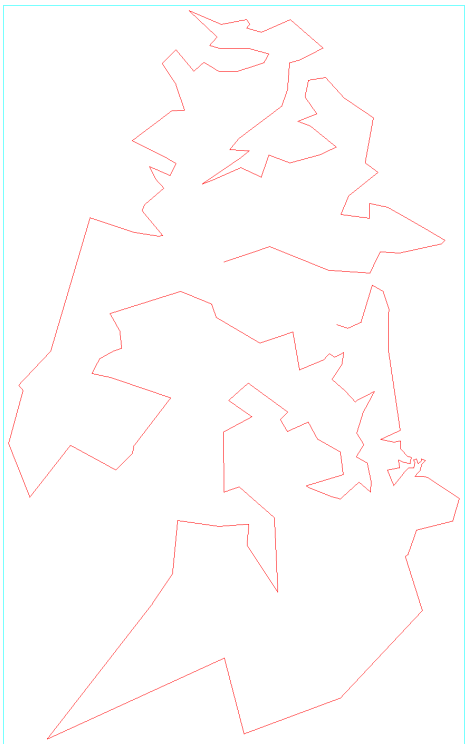
6.3.1 Path Length

The path length objective function was tested by solving a set of three standard travelling salesman problems obtained from TSPLIB. The global optimums have been identified for the three problems and were compared to the solutions generated by the genetic algorithm. Figures 6-8, 6-9 and 6-10 compare the optimal solution to the generated solution for the QA194, XIT1083 and FI10639 problems respectively.

The actual lengths of the paths for the optimal and generated solutions for each problem is shown in Table 6.2. The genetic algorithm was able to generate solutions to the QA194, XIT1083 and FI10693 problems within 4%, 5% and 4% respectively.

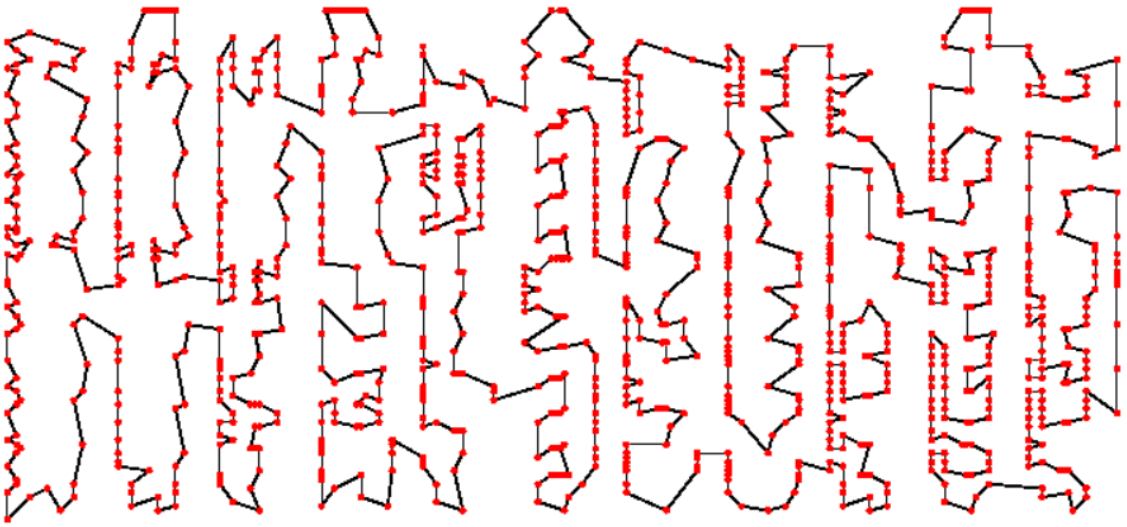


(a) Optimal solution to QA194.

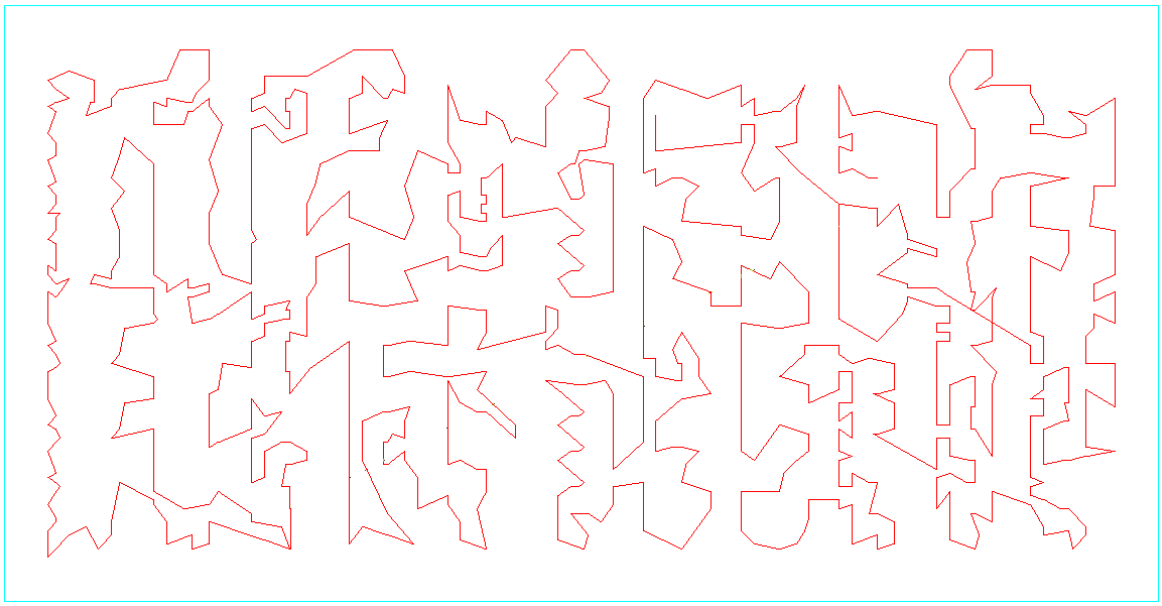


(b) Generated solution to QA194.

Figure 6-8: Comparison of optimal vs generated solutions to the QA194 TSP.



(a) Optimal solution to XIT1083.

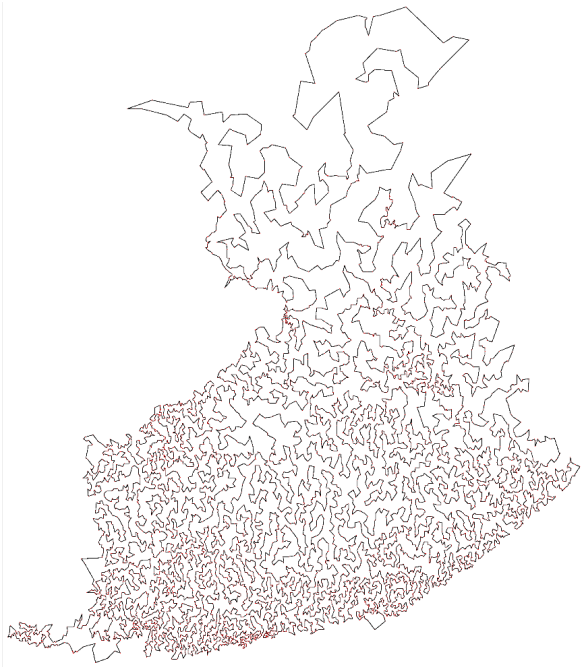


(b) Generated solution to XIT1083.

Figure 6-9: Comparison of optimal vs generated solutions to the XIT1083 TSP.

Table 6.2: Comparison of toolpaths generated by the genetic algorithm vs. optimal paths for three TSPs.

	Length		
TSP	Optimal	Generated	Fitness
QA194	9,352	9,737	0.96
XIT1083	3,558	3,740	0.95
FI10639	520,527	540,427	0.96



(a) Optimal solution to FI10639.

(b) Generated solution to FI10639.

Figure 6-10: Comparison of optimal vs generated solutions to the FI10639 TSP.

6.3.2 Path Straightness

The two path straightness objective functions were tested by obtaining the optimal path for a 10x10 grid and comparing the generated solutions to these paths. Both the turn based and angular based straightness objectives were tested. Figures 6-11-6-14 show the optimal and generated toolpaths for both objective functions.

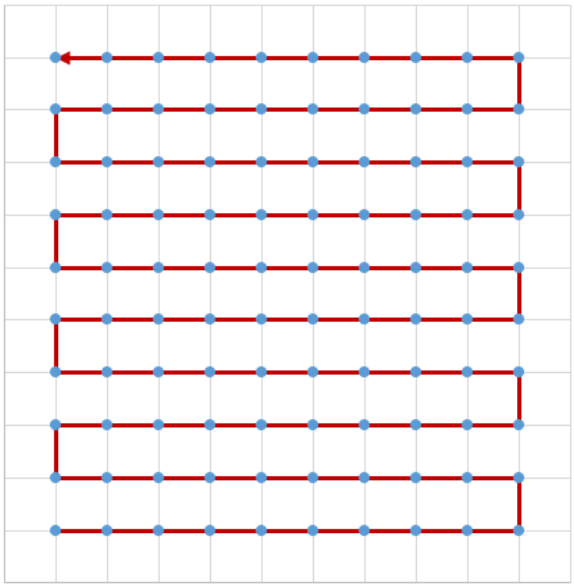


Figure 6-11: Optimal bidirectional toolpath for 10x10 grid.

The results of the test cases can be seen in Table 6.3. Both objective functions performed well with the bidirectional toolpaths having an average fitness within 2% and the spiral toolpaths having an average fitness within 1% of the optimal solutions.

Table 6.3: Comparison of generated vs optimal toolpaths for bidirectional and spiral strategies.

	Straightness		
TSP	Optimal	Generated(Average)	Fitness
Bidirectional	18 turns	18.33 turns	0.98
Spiral	1620deg	1636.8deg	0.99

The toolpath generation for the bidirectional objective function produced optimal solutions more often than for the spiral objective function.

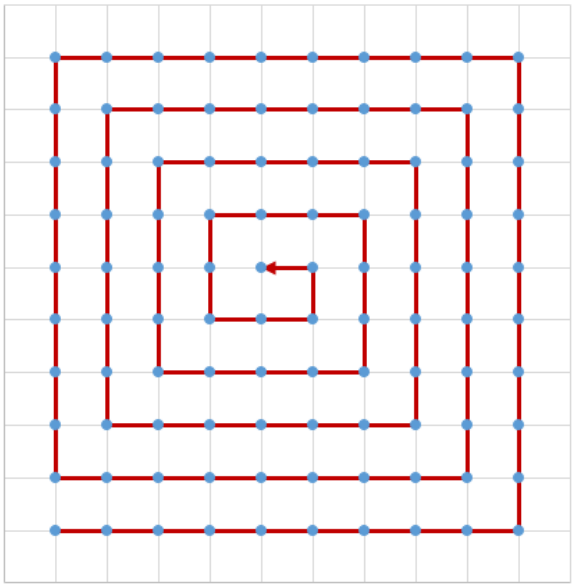


Figure 6-12: Optimal spiral toolpath for 10x10 grid.

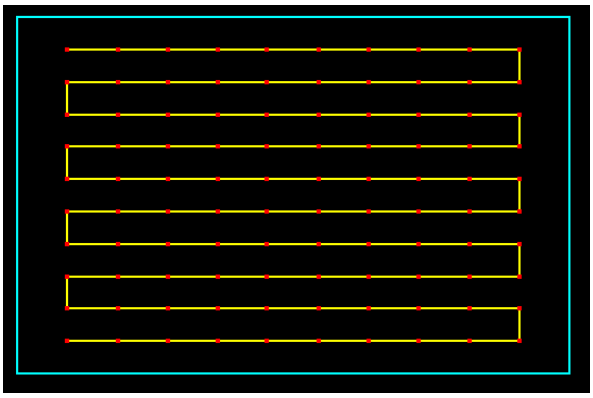


Figure 6-13: Generated bidirectional toolpath for 10x10 grid.

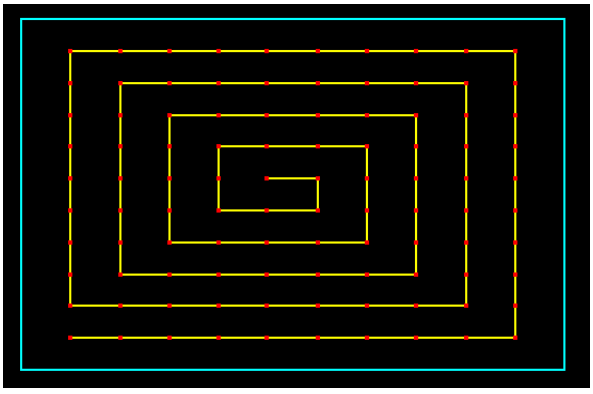


Figure 6-14: Generated spiral toolpath for 10x10 grid.

6.3.3 Tool Engagement

Tool engagement was tested by setting the intended tool engagement for the toolpath at 20%, 40%, 60%, 80% and 100% and then generating a toolpath for the fishhead part at each interval of tool engagement.

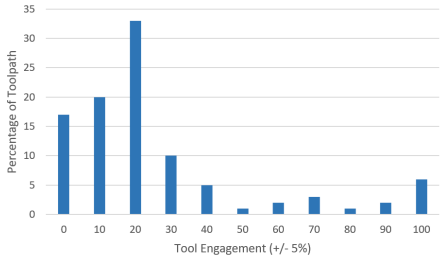
The five sub-figures in Figure 6-15 illustrate the percentage of tool engagement over the generated toolpath when the objective function is set to tool engagement values between 20% and 100% in increments of 20%. The histogram for each tool engagement setting show that the genetic algorithm was able to generate toolpaths where a majority of the toolpath had a tool engagement value equal to the value the objective function was set at. There was also a sharp drop-off in the amount of the toolpath machining at a tool engagement higher than the value in the objective function. Each histogram also shows a slight increase at the 100% tool engagement value which is due to the tool first entering the workpiece where 100% of the tool will be engaged.

6.3.4 Multi-Objective

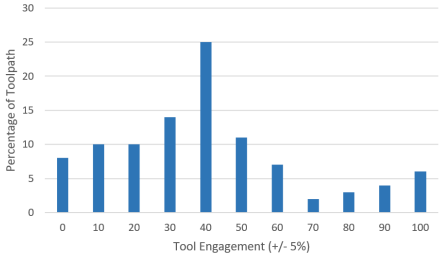
To test the performance of the multi-objective function, the multi-objective function was used to generate a toolpath for the fishhead test part and the three components of the multi-objective function were compared to the equivalent generated toolpath for each singular objective. The three objectives used were the path length, straightness (bidirectional) and tool engagement which was set to 50%. The generated toolpath using the multi-objective function can be seen in Figure 6-16.

Table 6.4 list the various metrics included in the multi-objective function and compares them to their respective metrics for the toolpath generated with that individual objective function.

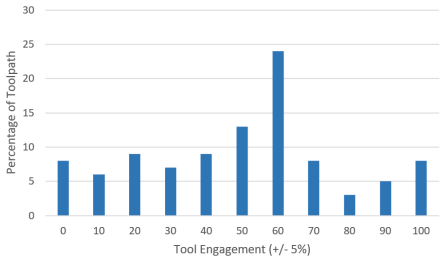
It can be seen from the data in Table 6.4 that the toolpaths generated using the multi-objective function performed almost as well in each metric when compared to the same metric produced with the individual objective function. The length and straightness performed within 2.6% and 2.4% of the solutions produced by the individual objective functions re-



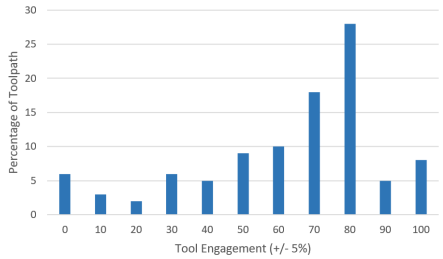
(a) Tool engagement across toolpath at 20% tool engagement.



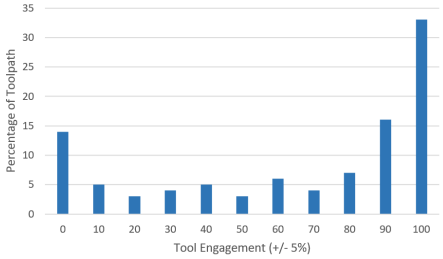
(b) Tool engagement across toolpath at 40% tool engagement.



(c) Tool engagement across toolpath at 60% tool engagement.



(d) Tool engagement across toolpath at 80% tool engagement.



(e) Tool engagement across toolpath at 100% tool engagement.

Figure 6-15: Tool engagement across toolpaths at various levels of tool engagement.

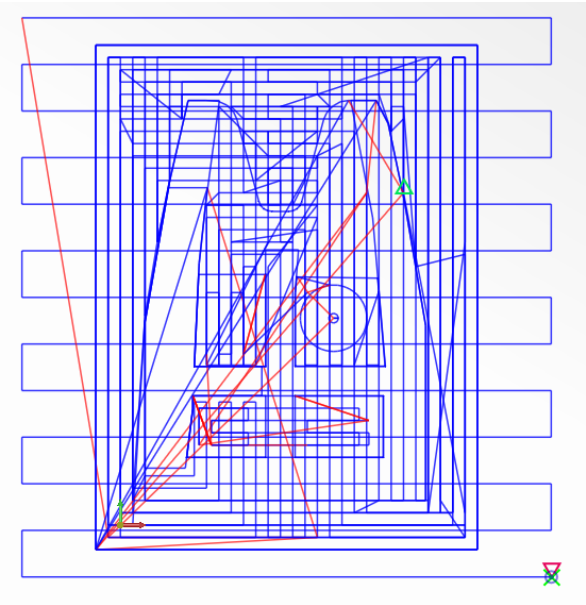
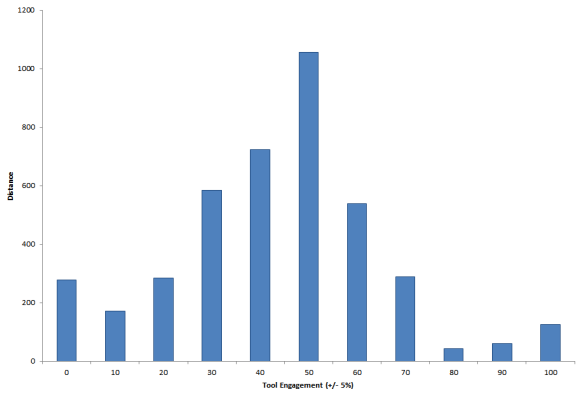


Figure 6-16: The generated toolpath using the multi-objective function.

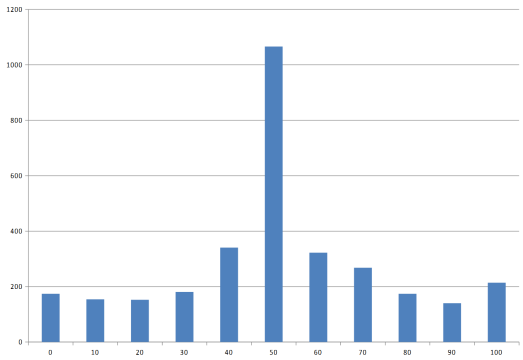
Table 6.4: Comparison of toolpath metrics between the multi-objective and single-objective fitness functions.

	Objective		
	Length	Turns	Time Taken
Length	2937	185	10s
Straightness	3615	41	10s
Engagement	3999	260	30s
Multiple	3012	42	30s

spectively. Figures 6-17a and 6-17b illustrate the tool engagement metrics over the generated toolpaths for the individual and multiple objective functions respectively.



(a) Tool engagement histogram for single objective toolpath generation.



(b) Tool engagement histogram for multiple objective toolpath generation.

Figure 6-17: Comparison of single vs multi-objective tool engagement over the toolpath.

The tool engagement values for the toolpath generated using the multi-objective were significantly better than the one's produced by the individual objective function. This is due to the nature of the tool engagement for a toolpath. The tool engagement for any point in the tool path is dependent on every point in the toolpath previous to it. This makes tool engagement a difficult objective to fully optimise as a slight change to an early portion of the toolpath can vastly alter the tool engagement for the rest of the toolpath. It seems that the other objectives included with the multi-objective function help optimise the tool engagement further than it can if only the single objective is used.

6.4 Comparison to Currently Used Methods

In this section the fishhead test part is used to compare the performance of the genetic algorithm to three widely used CAM packages to generate spiral and bidirectional toolpaths.

6.4.1 Toolpath for Test Parts Generated by the Developed Genetic Algorithm

The following figures show the toolpaths generated using the genetic algorithm for the fishhead part. The generated toolpath with the bidirectional machining strategy can be seen in Figure 6-18 and the spiral toolpath can be seen in Figure 6-19.

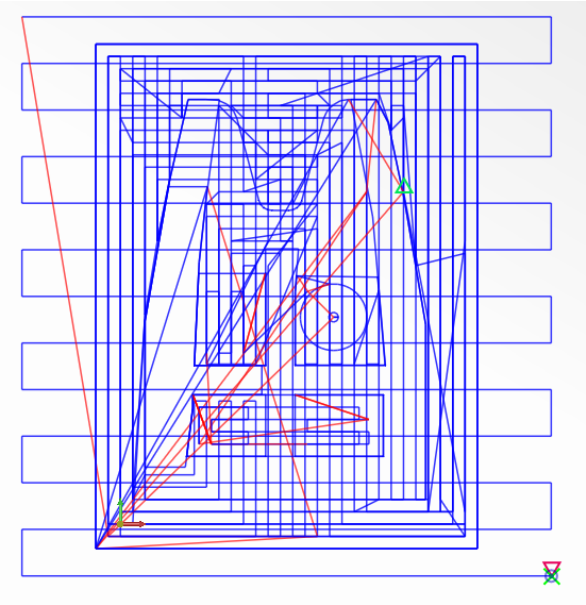


Figure 6-18: Bidirectional toolpath for the fishhead test part generated with the GA.

6.4.2 Toolpath for Test Parts Generated by CAM Software

The following figures show example toolpaths generated by a CAM package. The example toolpaths were generated using the FeatureCAM software package for the fishhead part. The generated toolpath with the bidirectional machining strategy can be seen in Figure 6-20 and the spiral toolpath can be seen in Figure 6-21.

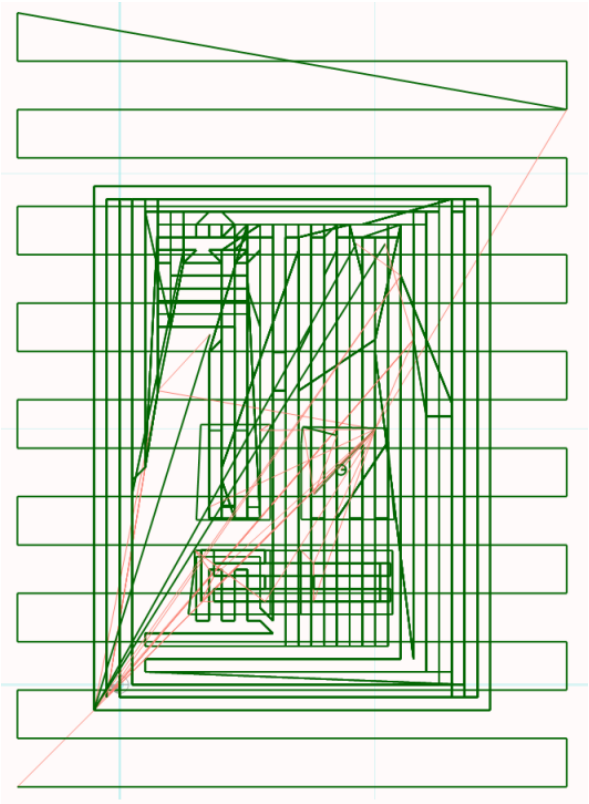


Figure 6-19: Spiral toolpath for the fishhead test part generated with the GA.

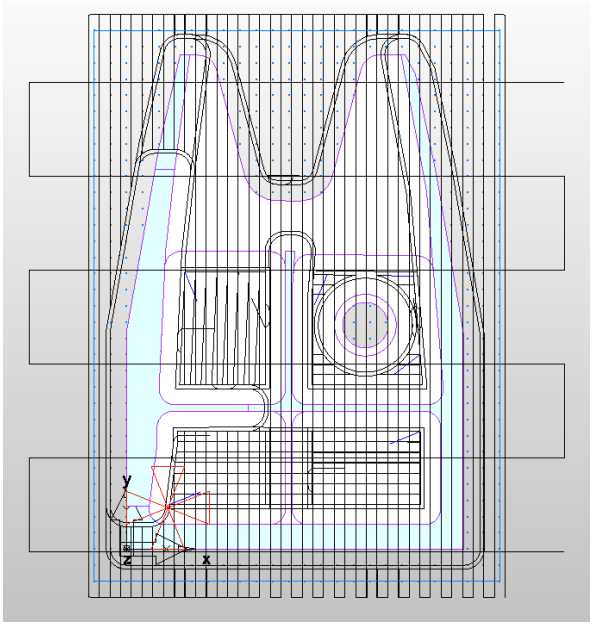


Figure 6-20: Bidirectional toolpath for the fishhead test part generated with FeatureCam.

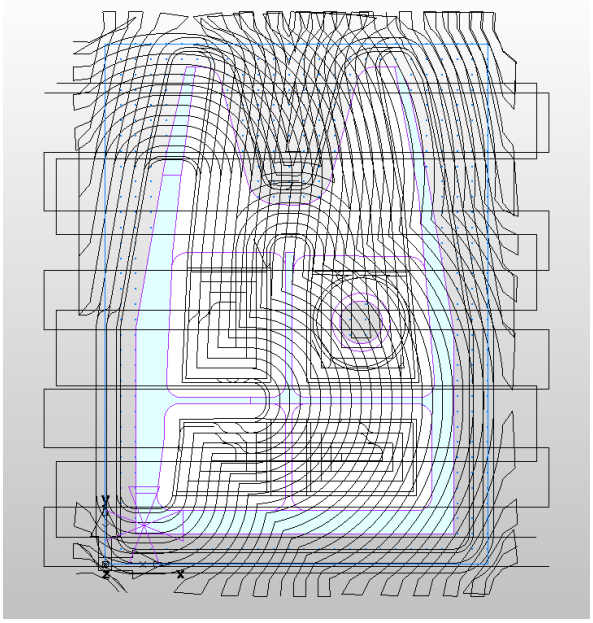


Figure 6-21: Spiral toolpath for the fishhead test part generated with FeatureCam.

6.4.3 Overall Comparison of GA Generated VS CAM Generated Toolpaths

The complete set of data for the generated toolpaths using the genetic algorithm and the three CAM packages is shown in Table 6.5. Each piece of software was used to generate a spiral and bidirectional toolpath. The total feed move distance, rapid move distance, overall distance, number of turns and turn angle for each toolpath is listed in Table 6.5.

Table 6.5: Comparison of toolpath metrics between CAM and GA generated toolpaths for the fishhead test part.

	Length(mm)			Straightness	
Software	Feed Move	Rapid Move	Total Move	Turns	Turn Angle
Bidirectional					
GA	25,282	2,395	27,729	423	-
NX	24,950	2,285	27,235	415	-
FeatureCam	24,395	2,325	26,720	412	-
MasterCam	25,030	2,525	27,555	431	-
Spiral					
GA	24,103	1,838	25,941	-	34686
NX	23,830	1,680	25,510	-	34090
FeatureCam	23,740	1,945	25,685	-	34358
MasterCam	24,010	1,795	25,805	-	35670

As can be seen in Table 6.5, the performance of the genetic algorithm was very similar to

the tested CAM packages. For the bidirectional toolpath, the GA performed within 3.6% of the best performing CAM package for length and 2.6% for number of turns. For the spiral toolpath, the GA performed within 3.1% of the best performing CAM package for length and 1.8% for total turn angle of the toolpath.

6.4.4 Comparison Against Car's Genetic Algorithm

As discussed in Chapter 5, the genetic algorithm developed in this research will be tested against a similar algorithm developed by Car. The results for the four test cases can be seen in Table 6.6. It can be seen that the genetic algorithm developed in this research outperformed the one developed by Car in all four test cases with respect to both fitness and time taken.

Table 6.6: Comparing the performance of the developed GA to Car's.

	Car's Algorithm		**Current Algorithm**	
Points	Fitness	Time (sec)	Fitness	Time (sec)
10x10	91.31	15.00	99.91	0.52
100x100	90.78	54.85	97.34	13.71
100x75	89.39	146.33	97.52	11.23
100x80	90.71	156.00	97.43	11.90

6.5 Summary

This chapter presents all of the results obtained from the test cases outlined in Chapter 5. The behaviour of the genetic algorithm was tested and it was found that the number of points used in the toolpath generation increased the number of generations required to converge on an optimal solution. This also means that the time required to converge on a solution increases. The quality of the generated solution only reduced to a minimum of 97.2% of the optimal solution at 1000 points and did not further reduce as the number of points increased.

Due to the stochastic nature of the genetic algorithm, the variation of the generated solutions was tested and the spread of fitness values for the generated solutions was no

more than 0.58%. The spread of fitness values also decreased as the number of generations increased.

The objective functions developed for the prototype were validated using a set of test cases. The four objective functions all behaved as intended by producing optimal or near optimal toolpaths for the respective objective that was being optimised. The two types of straightness objectives were successful in producing a toolpath with a bidirectional and spiral machining strategy. The multi-objective function generated a toolpath that contained metrics that were very similar to the ones produced by the three individual objective functions.

The genetic algorithm was able to generate toolpaths that were very similar in performance to the three tested CAM packages with respect to the length, number of turns and turn angle for both the bidirectional and spiral machining strategies. The genetic algorithm developed in this research also outperformed the similar algorithm produced by Car in all four test cases with respect to the quality of solution and the time taken to generate a solution.

Chapter 7

Discussion

7.1 Introduction

This chapter will discuss the issues identified and considered throughout the conducted research with respect to the research context and scope discussed in Chapter 1 Section 1.2.

7.2 State-of-the-art in Milling Toolpath Generation

A review of the literature on toolpath generation for CNC milling was performed in Chapter 2. A large number of techniques and algorithms were identified which generated CNC milling toolpaths for a wide variety of feature types. This large number of techniques and lack of a single overarching solution for toolpath generation emphasises the complexity of the toolpath generation problem. Although the individual analytical techniques can generate toolpaths quickly and efficiently, there is little consideration for the optimisation of the toolpath characteristics.

7.3 State-of-the-art in Methods for Solving the Travelling Salesman Problem

To design a framework to solve the adapted travelling salesman problem a review of the current methods for solving the traditional travelling salesman problem was performed in Chapter 3 Section 3.5. Many algorithms and techniques have been developed to solve the travelling salesman problem ranging from brute force to meta-heuristics. The review found that meta-heuristics were the most efficient tool in producing optimal to near optimal solutions to the travelling salesman problem.

7.4 A Novel Framework for Modelling Toolpath Generation as a Travelling Salesman Problem

The framework outlined in Chapter 3 provides a novel method for generating CNC machining toolpaths. The main concept of the framework is to have one method of generating toolpaths that is independent of geometry and that is flexible in such a way that the generated toolpaths adhere to a set of objectives as required by the user.

The framework consists of three main parts:

- Converting the toolpath generation problem for a given geometry into a travelling salesman problem that will produce a legal machining toolpath for all its possible solutions.
- Optimising the toolpath generated for the given geometry by finding more optimal solutions to the defined travelling salesman problem.
- Defining the objective functions used in optimising the generated toolpaths.

The proposed framework is an improvement over the currently used methods in the fact that it consolidates the necessity of maintaining a large number of distinct algorithms for generating toolpaths based on the geometry of the feature or any machining strategy that is

required. Due to the core component of the framework consisting of an optimisation technique, the generated toolpaths will in theory be the best possible toolpath for the objective that is being optimised. There is also the possibility of optimising various objectives simultaneously which is currently not possible with current methods.

7.4.1 Generating a TSP for a Given Geometry

It was identified in the research that a machining volume can be converted into a TSP by discretising the volume into a set of points. The TSP that is constructed will be an adaptation of the standard TSP due to the fact that any solution to this adapted TSP will have to be a legal machining toolpath. Chapter 4 identified all of alterations that were made to the standard TSP to produce machining toolpaths.

The first main adaptation is in the specific method that the volume is discretised so that if every point is visited in the point set by a given tool, all of the required material will be removed. The second main adaptation is the additional clauses that the TSP should adhere to.

7.4.2 Optimising the Generated TSP

Although every possible solution to the adapted TSP will produce a viable machining toolpath, the majority of solutions will be very inefficient at removing material. To ensure that the generated toolpath will be efficient, an optimal or near optimal solution to the TSP will have to found. This can be done by using any number of optimisation techniques.

7.4.3 Defining Objective Functions for Optimisation

An objective function must be defined to guide the progress of any optimisation technique. Chapter 4 identifies four common characteristics considered when generating machining toolpaths and analyses the process of developing the objective functions for these characteristics so that they can be used in any optimisation technique.

7.4.4 A Computational Platform Prototype of the Novel Framework

In order to determine the capabilities of the framework discussed in the previous section, a prototype implementation was developed in the form of a computational platform. The development of the prototype is described in Chapter 4. The entire computational platform was developed using Java to take advantage of the language's object oriented structure.

The prototype is capable of generating optimised machining toolpaths given in G-Code directly from a STEP-NC description of a part. This is done by using a STEP-NC interpreter to obtain the part geometry information which is then discretised into a layered grid of points to be used in the TSP. The optimisation technique developed for the prototype was a genetic algorithm. The solutions to the TSP are stored in the genetic information of the individuals. The genetic algorithm is allowed to progress until no further improvements are found.

The genetic algorithm proved to be an exceptional choice at maintaining the modular structure of the framework with respect to the objective functions. The characteristic to be optimised could be achieved by only altering the fitness function of the genetic algorithm. This results in the only user input to generating a machining toolpath from a STEP-NC description of the part being the specification of the objective function.

7.5 Evaluation of the Novel Framework and Computational Platform Using Test Cases

A set of test cases were designed in Chapter 5 to assess the computational platform developed in this research. The test cases can be split into two main categories: Identifying the behaviour of the optimisation technique used in the computational platform and the performance of the computational platform with respect to currently established methods of generating machining toolpaths.

The first set of test cases verified that the prototype implementation of the framework developed in this research was able to create a travelling salesman problem for a number of feature types and geometries. It was also able to produce optimal or near optimal solutions

to the created travelling salesman problem whilst optimising for four different individual objective functions and a multi-objective function containing three objectives. The test cases also demonstrated the capabilities of the framework to produce viable and optimal machining toolpaths with varying characteristics for any type of geometry by only altering the objective function.

The second set of test cases evaluated the performance of the developed computational platform with respect to three widely used CAM packages (NX, FeatureCAM and Master-CAM). One of the main differences observed from the test cases was that the length of rapid moves was much shorter for the toolpaths generated using the genetic algorithm than for the toolpaths generated using the CAM packages. This is due to the fact that one of the conditions of solving the travelling salesman problem is that the path through the set of points must be one continuous path. This results in the generation of toolpaths where the only rapid moves are between the end and start of each new feature as well as cut depths. The CAM generated toolpaths might contain rapid moves within each cut depth of an individual feature when avoiding feature boundaries or boss boundaries.

The time taken to perform rapid moves is more often than not negligible when compared to feed moves. Therefore the amount of time saved by reducing rapid moves is not substantial enough to be considered beneficial to the manufacture of a part.

However the genetic algorithm did perform very well when compared to the CAM packages with respect to the amount feed moves as well as the other two straightness characteristics measured of the toolpaths. In some of the test cases the genetic algorithm actually outperformed the CAM package and in the worst performing test case still the genetic algorithm still produced a toolpath within 3.6% of the best performing CAM software.

7.6 Limitations of the Computational Platform

The following is a list discusses the limitations of the framework proposed in this research:

- The tool used in machining any of the generated toolpaths from the framework is

assumed to be prismatic (e.g. an endmill) where the entire volume of material within the radius of the tool and the cut depth is removed at each individual point. For any tool that is not prismatic, it cannot be ensured that all of the required material will be removed by the tool if it visits every point of the TSP produced for a feature.

- The generation of grid points for a feature only functions properly if the feature contains a closed boundary as it would in the case of a closed pocket. For other feature types such as an open pocket or a facing operation the feature requires conversion to an equivalent closed pocket with slightly larger dimensions in the area containing the open boundary. This produces toolpaths that would contain larger portions of the toolpath not engaged with material than if the toolpath was generated using conventional methods.

- Due to the nature of the travelling salesman problem, the optimal path as well as its length is unknown before the optimisation method begins. Therefore it is very difficult to assess whether the solution found by any optimisation technique is a global or local optimum. This leads to the problem of deciding when to end the optimisation process.

- Individual objectives can be combined into a multi-objective function so long as the individual objectives are not conflicting. If two objectives are conflicting then it will be impossible to optimise one variable whilst not reducing the optimisation for the other.

Chapter 8

Conclusion and Future Work

8.1 Introduction

This chapter will cover the conclusions obtained from this research along with the overall contribution to knowledge. Potential areas in which this research could be expanded and further investigated is covered in the final section of this chapter.

8.2 Conclusions

- Current methods of toolpath generation vary greatly depending on the geometry, machining strategy or objective being optimised to. These current methods also do not have the capability of generating optimised toolpaths with respect to more than one objective.

- It is possible to model toolpath generation as an adapted travelling salesman problem by discretising the volume of material being removed through machining into a uniform set of grid points and performing various optimisation techniques on these points.

- The travelling salesman problem is an incredibly difficult optimisation problem to solve due to its large solution space at large number of points. It was found that meta-

heuristics are the current most efficient method of producing optimal and near optimal solutions to the travelling salesman problem.

- The framework developed in this research allows for a CNC machining toolpath to be generated from a part described in the STEP-NC format by solving an adapted travelling salesman problem using an optimisation algorithm with a series of objective functions particular to machining toolpaths.
- Toolpath characteristics and machining strategies can be incorporated into the generated toolpaths by adjusting the objective function that the optimisation algorithm adheres to. This allows for a flexible algorithm which can generate a wide array of optimised toolpaths whilst also being independent of part geometry.
- The framework allows for one or more objective functions to be optimised at once. This enables the optimisation algorithm to generate toolpaths which are optimal with respect to multiple objectives allowing for the generation of more complex and useful toolpaths.
- A prototype of the framework proposed in this research has been realised in the form of a computational platform. A genetic algorithm was chosen as the optimisation algorithm to be used in solving the travelling salesman problem. The results of the development was a modular, flexible system which was able to generate a machining toolpath for a part independent of its geometry whilst maintaining the ability to switch objective functions without altering the rest of the system.
- A test part which was designed based off of an industrially used aerospace component was used as a test case in evaluating the prototype's ability with respect to currently used computational platforms in generating machining toolpaths. The prototype performed within 3.6% of solutions provided by the computational platforms with respect to path length and straightness.
- The framework proposed in this research can easily be adapted and expanded to account for further developments in the field of CNC machining.

8.3 Contribution to Knowledge

The main contribution to knowledge of this research is in the novel interpretation of CNC toolpath generation and optimisation as a travelling salesman problem and the framework in which to achieve this. The framework allows for a flexible method in which to generate toolpaths and optimise them with respect to one or more objectives independent of part geometry. This allows for toolpaths to be generated for a wide array of feature types with only one core optimisation algorithm which can be easily adapted to the needs of the individual. The proposed computational platform can generate toolpaths for a part described using STEP-NC in the form of G code to be used on a CNC machine. This allows parts described in the newer machine control language STEP-NC to be used on CNC machines using the older machine control language of G-code which is a valuable contribution to this field of research.

8.4 Future Work

8.4.1 Including Additional Toolpath Characteristics

This research focused on the optimisation of toolpaths with respect to three toolpath characteristics (length, straightness and tool engagement) as well as adhering to two types of machining strategies (bidirectional and spiral). There are for more toolpath characteristics and machining strategies that could be considered and included in the objective function. The following is a list of other toolpath characteristics that could be considered:

- Cutting force
- Energy usage
- Tool wear
- Surface quality
- Thin walls

- Climb/Conventional milling
- Machining strategies(e.g. Contour, trochoidal)

8.4.2 5-Axis Machining

The framework developed in this research could be expanded to handle 5-axis machining toolpaths. The current method uses a genetic algorithm that creates individuals with a genetic sequence that corresponds to coordinate positions of the tool. In this method it is assumed that the tool is always parallel to the Z-axis, which would be the case with traditional 3-axis machines where the tool position can only be translated in the X, Y or Z axes. For machines that incorporate one or more rotary axes, the angle of the tool tip with respect to the part can be adjusted.

To adapt the current framework to be utilised for 5-axis machining toolpaths, the genetic structure of the individuals would require some adjustment. Each gene would consist of multiple components as opposed to the current single parameter of the gene. The gene information would be made up of a position component and two angle components to fully describe the tool tip position and orientation at each grid point. An example of a multi-component gene can be seen in Figure 8-1.

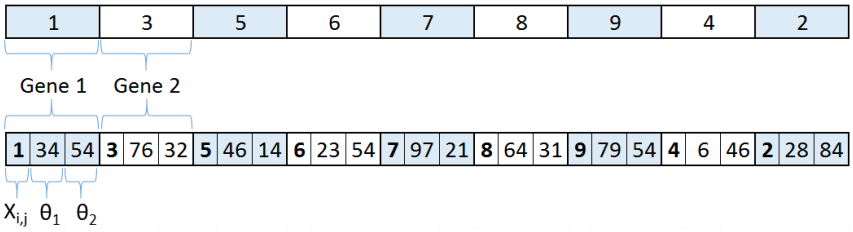


Figure 8-1: Comparison of single component and multi-component genes.

No additional genes would be required to describe a system in 5-axis compared to 3-axis, only the extra components of the gene. $X_{i,j}$, θ_1 and θ_2 in Figure 8-1 correspond to a grid point, angle of first rotary axis and angle of second rotary axis respectively.

The fitness function of the genetic algorithm would require a new component to calculate the fitness of the new gene components. However the remaining modules of the genetic

algorithm can remain the same and still successfully create new individuals from existing ones to obtain an optimal solution.

As with any optimisation algorithm, the more variables that are introduced into the system, the more effort that is required to find an optimal solution. Therefore it is possible that more efficient operators will have to be developed to efficiently reduce the now larger solution space.

8.4.3 Additive Manufacturing Toolpaths

Another area of research that could benefit from the framework developed in this research is that of additive manufacturing. More specifically the generation of toolpaths or laser rastering paths used in additive manufacturing. The method of generating a process plan for an additively manufactured part share many similarities regardless of the specific technology being used. A typical process in producing an additive part can be described as follows:

- Obtain model of part to be manufactured.
- Decide on a build direction for material to be deposited.
- Slice model into layers of specified thickness.
- Decide on the infill density and pattern.
- Generate paths for the tool/laser to follow for each layer.
- Convert paths into machine code instructions and run the code on the machine.
- Perform any necessary post-processing on part.

Steps 3, 4 and 5 are very similar to what is required in the framework developed in this research. With some adjustments the framework could produce toolpaths that would be optimised to be used with an additive process instead of a subtractive one. One of the current drawbacks of additive processes is the time required to deposit all the necessary material. The framework from this research could potentially produce optimised paths with respect to

length to reduce the time taken to manufacture additive parts. Other objective functions could be developed which are exclusive to additive manufacturing as they are identified through ongoing research in the field.

8.4.4 Quantum Computer Toolpath Generation

Quantum computing is an exciting new area of research which could have a large potential impact on solving optimisation problems. In classical computing, information is stored in bits where each bit can store either the value of 1 or 0. Therefore when solving optimisation problems, the computational algorithm has to iterate through most of the permutations of bit values and evaluate the result of each possible solution. For most optimisation problems there is a very large number of solution states and using the problem tackled in this research as an example it can be seen that this is very time consuming.

In quantum computing the information is stored in "Qubits", which can also store a value of 1 or 0 but unlike a regular bit it can store a superposition of both states[167]. This would theoretically allow all of the various states of qubits to be evaluated simultaneously instead of iteratively. Quantum computing could vastly reduce the computational effort and time required to solve optimisation problems such as the travelling salesman problem.

Some research has already started on developing the algorithms required to solve optimisation problems using a quantum computer. Han developed a genetic quantum algorithm to be used in solving combinatorial optimisation problems [168, 169]. Goswami developed a framework for solving travelling salesman problems using quantum computing models[170]. It is still unknown whether a true quantum computer will be achievable and if so, whether the theoretical performance advantages of solving optimisation problems will be realisable in practice.

References

[1] F. Jovane, Y. Koren, and C. Boer, "Present and future of flexible automation: towards new paradigms," *CIRP Annals-Manufacturing Technology*, vol. 52, no. 2, pp. 543–560, 2003.

[2] H. A. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *International journal of flexible manufacturing systems*, vol. 17, no. 4, pp. 261–276, 2005.

[3] K. Preiss and E. Kaplansky, "Automated part programming for CNC milling by artificial intelligence techniques," *Journal of Manufacturing Systems*, vol. 4, no. 1, pp. 51–63, 1985.

[4] R. Licari, E. L. Valvo, and M. Piacentini, "Part program automatic check for three axis CNC machines," *Journal of Materials Processing Technology*, vol. 109, no. 3, pp. 290–293, 2001.

[5] M. Hardwick, Y. F. Zhao, F. M. Proctor, A. Nassehi, X. Xu, S. Venkatesh, D. Odendahl, L. Xu, M. Hedlind, M. Lundgren *et al.*, "A roadmap for step-nc-enabled interoperable manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 68, no. 5-8, pp. 1023–1037, 2013.

[6] ISO 14649-10:2004, *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data.* Geneva, Switzerland: ISO, Geneva, Switzerland.

[7] J. E. Bobrow, "NC machine tool path generation from CSG part representations," *Computer-Aided Design*, vol. 17, no. 2, pp. 69–76, 1985.

[8] A. G. Requicha, "Representations for rigid solids: Theory, methods, and systems," *ACM Computing Surveys (CSUR)*, vol. 12, no. 4, pp. 437–464, 1980.

[9] A. Jacobson. CSG operations in libigl. [Online]. Available: http://alecjacobson.com/weblog/media/cube-sphere-cylinders-csg-tree.jpg

[10] J. Zhao, Y. Ohnishi, G. Zhao, and T. Sasaki, *Advances in Discontinuous Numerical Methods and Applications in Geomechanics and Geoengineering*, ser. Proceedings and Monographs in Engineering, Water and Earth Sciences. Taylor & Francis, 2012. [Online]. Available: http://books.google.co.uk/books?id=L-FqkNuejjUC

[11] I. C. Braid, "The synthesis of solids bounded by many faces," *Communications of the ACM*, vol. 18, no. 4, pp. 209–216, 1975.

[12] I. Stroud and H. Nagy, *Solid Modelling and CAD Systems: How to Survive a CAD System*. Springer, 2011.

[13] V. Chandru, S. Manohar, and C. E. Prakash, "Voxel-based modeling for layered manufacturing," *Computer Graphics and Applications, IEEE*, vol. 15, no. 6, pp. 42–47, 1995.

[14] S. Patil and B. Ravi, "Voxel-based representation, display and thickness analysis of intricate shapes," in *Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on*. IEEE, 2005, pp. 6–pp.

[15] K. Chui, K. Yu, and T. Lee, "Direct tool-path generation from massive point input," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 216, no. 2, pp. 199–206, 2002.

[16] A. C. Lin and H. T. Liu, "Automatic generation of NC cutter path from massive data points," *Computer-Aided Design*, vol. 30, no. 1, pp. 77–90, 1998.

[17] D. Dragomatz and S. Mann, "A classified bibliography of literature on NC milling path generation," *Computer-Aided Design*, vol. 29, no. 3, pp. 239–247, 1997.

[18] G. Elber and E. Cohen, "Toolpath generation for freeform surface models," *Computer-Aided Design*, vol. 26, no. 6, pp. 490–496, 1994.

[19] M. Held, G. Lukacs, and L. Andor, "Pocket machining based on contour-parallel tool paths generated by means of proximity maps," *Computer-Aided Design*, vol. 26, no. 3, pp. 189–203, 1994.

[20] J. Jeong and K. Kim, "Tool path generation for machining free-form pockets using Voronoi diagrams," *The International Journal of Advanced manufacturing Technology*, vol. 14, no. 12, pp. 876–881, 1998.

[21] H. Persson, "NC machining of arbitrarily shaped pockets," *Computer-Aided Design*, vol. 10, no. 3, pp. 169–174, 1978.

[22] H. Qiu, C. Kai, and L. Yan, "Optimal circular arc interpolation for NC tool path generation in curve contour manufacturing," *Computer-Aided Design*, vol. 29, no. 11, pp. 751–760, 1997.

[23] M. K. Yeung and D. J. Walton, "Curve fitting with arc splines for NC toolpath generation," *Computer-Aided Design*, vol. 26, no. 11, pp. 845–849, 1994.

[24] E. Arkin, M. Held, and C. Smith, "Optimization problems related to zigzag pocket machining," *Algorithmica*, vol. 26, no. 2, pp. 197–236, 2000.

[25] S. C. Park and B. K. Choi, "Tool-path planning for direction-parallel area milling," *Computer-Aided Design*, vol. 32, no. 1, pp. 17–25, 2000.

[26] W. Lee and Y. B. Bang, "Design and implementation of an ISO14649-compliant CNC milling machine," *International Journal of Production Research*, vol. 41, no. 13, pp. 3007–3017, 2003.

[27] S. P. Radzevich, "Conditions of proper sculptured surface machining," *Computer-Aided Design*, vol. 34, no. 10, pp. 727–740, 2002.

[28] B. K. Choi, D. H. Kim, and R. B. Jerard, "C-space approach to tool-path generation for die and mould machining," *Computer-Aided Design*, vol. 29, no. 9, pp. 657–669, 1997.

[29] K. Chui, W. Chiu, and K. Yu, "Direct 5-axis tool-path generation from point cloud input using 3d biarc fitting," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 2, pp. 270–286, 2008.

[30] L. Chih-Ching, "A new approach to CNC tool path generation," *Computer-Aided Design*, vol. 30, no. 8, pp. 649–655, 1998.

[31] S. Ding, M. Mannan, A. N. Poo, D. Yang, and Z. Han, "Adaptive iso-planar tool path generation for machining of free-form surfaces," *Computer-Aided Design*, vol. 35, no. 2, pp. 141–153, 2003.

[32] H. Y. Feng and H. Li, "Constant scallop-height tool path generation for three-axis sculptured surface machining," *Computer-Aided Design*, vol. 34, no. 9, pp. 647–654, 2002.

[33] P. Wu, H. Suzuki, and K. Kase, "Three-axis NC cutter path generation for subdivision surface with Z-map," *JSME International Journal Series C*, vol. 48, no. 4, pp. 757–762, 2005.

[34] J. Zhu, T. Tanaka, and Y. Saito, "A rough cutting model generation algorithm based on multi-resolution mesh for sculptured surface machining," *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, vol. 1, no. 5, pp. 628–639, 2007.

[35] I. Lazoglu, C. Manav, and Y. Murtezaoglu, "Tool path optimization for free form surface machining," *CIRP Annals-Manufacturing Technology*, vol. 58, no. 1, pp. 101–104, 2009.

[36] S. G. Lee, H. C. Kim, and M. Y. Yang, "Mesh-based tool path generation for constant scallop-height machining," *The International Journal of Advanced Manufacturing Technology*, vol. 37, no. 1-2, pp. 15–22, 2008.

[37] ISO 10303-21:2002, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure.* Geneva, Switzerland: ISO, Geneva, Switzerland.

[38] ISO 14649-11:2004, *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling.* Geneva, Switzerland: ISO, Geneva, Switzerland.

[39] A. Lasemi, D. Xue, and P. Gu, "Recent development in CNC machining of freeform surfaces: A state-of-the-art review," *Computer-Aided Design*, vol. 42, no. 7, pp. 641–654, 2010.

[40] S. S. Makhanov, "Adaptable geometric patterns for five-axis machining: A survey," *The International Journal of Advanced Manufacturing Technology*, vol. 47, no. 9-12, pp. 1167–1208, 2010.

[41] Y. Takeuchi and T. Watanabe, "Generation of 5-axis control collision-free tool path and postprocessing for NC data," *CIRP Annals-Manufacturing Technology*, vol. 41, no. 1, pp. 539–542, 1992.

[42] B. Choi, J. Park, and C. Jun, "Cutter-location data optimization in 5-axis surface machining," *Computer-Aided Design*, vol. 25, no. 6, pp. 377–386, 1993.

[43] S. X. Li and R. B. Jerard, "5-axis machining of sculptured surfaces with a flat-end cutter," *Computer-Aided Design*, vol. 26, no. 3, pp. 165–178, 1994.

[44] J. Pi, E. Red, and G. Jensen, "Grind-free tool path generation for five-axis surface machining," *Computer Integrated Manufacturing Systems*, vol. 11, no. 4, pp. 337–350, 1998.

[45] K. Morishige, Y. Takeuchi, and K. Kase, "Tool path generation using C-space for 5-axis control machining," *Journal of manufacturing science and engineering*, vol. 121, no. 1, pp. 144–149, 1999.

[46] E. Bohez, S. S. Makhanov, and K. Sonthipermpoon, "Adaptive nonlinear tool path optimization for five-axis machining," *International Journal of Production Research*, vol. 38, no. 17, pp. 4329–4343, 2000.

[47] C. S. Jun, K. Cha, and Y. S. Lee, "Optimizing tool orientations for 5-axis machining by configuration-space search method," *Computer-Aided Design*, vol. 35, no. 6, pp. 549–566, 2003.

[48] S. S. Makhanov and S. A. Ivanenko, "Grid generation as applied to optimize cutting operations of the five-axis milling machine," *Applied numerical mathematics*, vol. 46, no. 3, pp. 331–351, 2003.

[49] C. Toh, "A study of the effects of cutter path strategies and orientations in milling," *Journal of materials processing technology*, vol. 152, no. 3, pp. 346–356, 2004.

[50] T. Chen, P. Ye, and J. Wang, "Local interference detection and avoidance in five-axis NC machining of sculptured surfaces," *The International Journal of Advanced Manufacturing Technology*, vol. 25, no. 3-4, pp. 343–349, 2005.

[51] C. Tournier and E. Duc, "Iso-scallop tool path generation in 5-axis milling," *The International Journal of Advanced Manufacturing Technology*, vol. 25, no. 9-10, pp. 867–875, 2005.

[52] C. J. Lu and K. L. Ting, "Subdivision surface-based finish machining," *International journal of production research*, vol. 44, no. 12, pp. 2445–2463, 2006.

[53] W. Anotaipaiboon and S. S. Makhanov, "Curvilinear space-filling curves for five-axis machining," *Computer-Aided Design*, vol. 40, no. 3, pp. 350–367, 2008.

[54] S. Lavernhe, C. Tournier, and C. Lartigue, "Optimization of 5-axis high-speed machining using a surface based approach," *Computer-Aided Design*, vol. 40, no. 10, pp. 1015–1023, 2008.

[55] W. He, M. Lei, and H. Bin, "Iso-parametric CNC tool path optimization based on adaptive grid generation," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 5-6, pp. 538–548, 2009.

[56] F. Ren, Y. Sun, and D. Guo, "Combined reparameterization-based spiral toolpath generation for five-axis sculptured surface machining," *The international journal of advanced manufacturing technology*, vol. 40, no. 7-8, pp. 760–768, 2009.

[57] Z. Haranud, Z. Jiang, T. Tanaka, and Y. Saito, "Optimal tool path generation method for freeform surface machining," in *Proceedings of the 5th International Conference on Leading Edge Manufacturing in 21st Century (LEM21)*, no. 09-207. The Japan Society of Mechanical Engineers (JSME), 2009, pp. 3–8.

[58] W. Anotaipaiboon and S. S. Makhanov, "Optimal grids for five-axis machining," *Mathematics and Computers in Simulation*, vol. 81, no. 3, pp. 636–655, 2010.

[59] A. Can and A. Ünüvar, "A novel iso-scallop tool-path generation for efficient five-axis machining of free-form surfaces," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 9-12, pp. 1083–1098, 2010.

[60] A. Lasemi, D. Xue, and P. Gu, "A freeform surface manufacturing approach by integration of inspection and tool path generation," *International Journal of Production Research*, vol. 50, no. 23, pp. 6709–6725, 2012.

[61] W. Anotaipaiboon and S. S. Makhanov, "Tool path generation for five-axis NC machining using adaptive space-filling curves," *International Journal of Production Research*, vol. 43, no. 8, pp. 1643–1665, 2005.

[62] E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys, *The traveling salesman problem: a guided tour of combinatorial optimization*. Wiley Chichester, 1985, vol. 3.

[63] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[64] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang, "The asymmetric traveling salesman problem: Algorithms, instance generators, and tests," in *Algorithm Engineering and Experimentation*. Springer, 2001, pp. 32–59.

[65] R. Kumar and H. Li, "On asymmetric tsp: Transformation to symmetric tsp and performance bound."

[66] F. Della Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 15–24, 1995.

[67] W. J. Wang, S. S. Lu, and C. F. Hsu, "Experiments on the position control of a one-link flexible robot arm," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 3, pp. 373–377, 1989.

[68] M. Gendreau, G. Laporte, and D. Vigo, "Heuristics for the traveling salesman problem with pickup and delivery," *Computers & Operations Research*, vol. 26, no. 7, pp. 699–714, 1999.

[69] T. Maekawa, "Computation of shortest paths on free-form parametric surfaces," *Transactions of the ASME-R-Journal of Mechanical Design*, vol. 118, no. 4, pp. 499–508, 1996.

[70] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: a survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[71] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, vol. 1, pp. 215–310, 1997.

[72] C. Chauhan, R. Gupta, and K. Pathak, "Survey of methods of solving tsp along with its implementation using dynamic programming approach," *International Journal of Computer Applications*, vol. 52, no. 4, pp. 12–19, 2012.

[73] Y. Jiang, T. Weise, J. Lassig, R. Chiong, and R. Athauda, "Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the tsp," in *Computational Intelligence in Production and Logistics Systems (CIPLS), 2014 IEEE Symposium on*. IEEE, 2014, pp. 148–155.

[74] A. R. Saiyed, "The traveling salesman problem," 2012.

[75] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. Rosamond, S. Saurabh, and Y. Villanger, "Local search: Is brute-force avoidable?" *Journal of Computer and System Sciences*, vol. 78, no. 3, pp. 707–719, 2012.

[76] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[77] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[78] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An algorithm for the traveling salesman problem," *Operations research*, vol. 11, no. 6, pp. 972–989, 1963.

[79] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 105–132.

[80] W. Zhang, "Truncated branch-and-bound: A case study on the asymmetric TSP," in *Proc. of AAAI 1993 Spring Symposium on AI and NP-Hard Problems*, 1993, pp. 160–166.

[81] D. Miller and J. Pekny, "Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem," *Operations Research Letters*, vol. 8, no. 3, pp. 129–135, 1989.

[82] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Operations Research Letters*, vol. 6, no. 1, pp. 1–7, 1987.

[83] J. Pearl, "Heuristics: Intelligent search strategies for computer problem solving," 1984.

[84] P. JUDEA, "Heuristics: intelligent search strategies for computer problem solving," 1985.

[85] R. L. Karg and G. L. Thompson, "A heuristic approach to solving travelling salesman problems," *Management science*, vol. 10, no. 2, pp. 225–248, 1964.

[86] A. Frieze, "Worst-case analysis of algorithms for travelling salesman problems," *Methods of Operations Research*, vol. 32, pp. 97–112, 1979.

[87] G. u. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[88] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," DTIC Document, Tech. Rep., 1976.

[89] D. S. Johnson and L. A. McGeoch, "Experimental analysis of heuristics for the stsp," in *The traveling salesman problem and its variations*. Springer, 2007, pp. 369–443.

[90] G. A. Croes, "A method for solving traveling-salesman problems," *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.

[91] F. Bock, "An algorithm for solving travelling-salesman and related network optimization problems," in *Operations Research*, vol. 6, no. 6. INST OPERATIONS RESEARCH MANAGEMENT SCIENCES 901 ELKRIDGE LANDING RD, STE 400, LINTHICUM HTS, MD 21090-2909, 1958, pp. 897–897.

[92] J. J. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA Journal on computing*, vol. 4, no. 4, pp. 387–411, 1992.

[93] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[94] D. Johnson, J. Bentley, L. McGeoch, and E. Rothberg, "Near-optimal solutions to very large traveling salesman problems," *Monograph, to appear*, 1987.

[95] C. H. Papadimitriou, "The complexity of the lin-kernighan heuristic for the traveling salesman problem," *SIAM Journal on Computing*, vol. 21, no. 3, pp. 450–465, 1992.

[96] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[97] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing: an international journal*, vol. 8, no. 2, pp. 239–287, 2009.

[98] C. Nilsson, "Heuristics for the traveling salesman problem," Tech. Rep.

[99] G. Renner and A. Ekárt, "Genetic algorithms in computer aided design," *Computer-Aided Design*, vol. 35, no. 8, pp. 709–726, 2003.

[100] F. Alex, "Simulation of genetic systems by automatic digital computers. i. introduction," *Aust. J. Biol. Sci*, vol. 10, pp. 484–491, 1957.

[101] A. Fraser, "Simulation of genetic systems by automatic digital computers. ii: Effects of unkage on rates under selection," *Austral. J. Biol. Sci*, vol. 10, pp. 492–499, 1957.

[102] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975. [Online]. Available: http://books.google.co.uk/books?id=YE5RAAAAMAAJ

[103] R. Brady, "Optimization strategies gleaned from biological evolution," *Nature*, vol. 317, no. 6040, pp. 804–806, 1985.

[104] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," 1995.

[105] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, no. 3, pp. 193–212, 1995.

[106] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," 1989.

[107] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, vol. 7, no. 1, pp. 65–85, 1988.

[108] J. J. Grefenstette and J. E. Baker, "How genetic algorithms work: A critical look at implicit parallelism," in *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers Inc., 1989, pp. 20–27.

[109] L. D. Whitley *et al.*, "The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best." in *ICGA*, 1989, pp. 116–123.

[110] H. Sengoku and I. Yoshihara, "A fast TSP solver using GA on JAVA," in *Third International Symposium on Artificial Life, and Robotics (AROB III98)*, 1998, pp. 283–288.

[111] V. M. Kureichick, V. V. Miagkikh, and A. P. Topchy, "Genetic algorithm for solution of the traveling salesman problem with new features against premature convergence," *TSURE Journal of Engineering*, 1996.

[112] P. Moscato, "On genetic crossover operators for relative order preservation," *C3P Report*, vol. 778, 1989.

[113] G. Üçoluk, "Genetic algorithm solution of the TSP avoiding special crossover and mutation," *Intelligent Automation & Soft Computing*, vol. 8, no. 3, pp. 265–272, 2002.

[114] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[115] M. Lidd, "Traveling salesman problem domain application of a fundamentally new approach to utilizing genetic algorithms," *Research sponsored in part by Air Force Office of Scientific Research and Office of Naval Research, Contract F4920-90-G-0033*, 1991.

[116] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the international joint conference on artificial intelligence*, vol. 1. Los Angeles, CA, USA, 1985, pp. 161–163.

[117] G. Syswerda, "Schedule optimization using genetic algorithms," *Handbook of genetic algorithms*, pp. 332–349, 1991.

[118] K. Deep and H. Mebrahtu, "New variations of order crossover for travelling salesman problem," *International Journal*, vol. 2, 2011.

[119] D. E. Goldberg and R. Lingle Jr, "Alleles, loci, and the traveling salesman problem," in *Proceedings of the 1st international conference on genetic algorithms*. L. Erlbaum Associates Inc., 1985, pp. 154–159.

[120] L. D. Whitley, T. Starkweather, and D. Fuquay, *Scheduling problems and traveling salesmen: The genetic edge recombination operator*. Colorado State University, Department of Computer Science, 1989.

[121] I. Oliver, D. Smith, and J. R. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*. L. Erlbaum Associates Inc., 1987, pp. 224–230.

[122] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," *Genetic algorithms and simulated annealing*, vol. 4, pp. 42–60, 1987.

[123] Y. Nagata, "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem," in *Proc. 7th ICGA*, 1997, pp. 450–457.

[124] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization. parallelism, learning," *Evolution, Springer-Verlag*, pp. 398–406, 1989.

[125] P. Larrañaga, C. M. Kuijpers, M. Poza, and R. H. Murga, "Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms," *Statistics and Computing*, vol. 7, no. 1, pp. 19–34, 1997.

[126] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the 1st international conference on genetic algorithms*. L. Erlbaum Associates Inc., 1985, pp. 160–168.

[127] K. Rani and V. Kumar, "Solving travelling salesman problem using genetic algorithm based on heuristic crossover and mutation operator," *International Journal of Research in Engineering and Technology*, vol. 2, no. 2, pp. 27–34, 2014.

[128] O. Abdoun and J. Abouchabaka, "A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem," *arXiv preprint arXiv:1203.3097*, 2012.

[129] H. S. Yoon and B. R. Moon, "An empirical study on the synergy of multiple crossover operators," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 212–223, 2002.

[130] J. Andre, P. Siarry, and T. Dognon, "An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization," *Advances in engineering software*, vol. 32, no. 1, pp. 49–60, 2001.

[131] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*. springer, 1998.

[132] W. Banzhaf, "The molecular traveling salesman," *Biological Cybernetics*, vol. 64, no. 1, pp. 7–14, 1990.

[133] D. B. Fogel, "An evolutionary approach to the traveling salesman problem," *Biological Cybernetics*, vol. 60, no. 2, pp. 139–144, 1988.

[134] D. B. Fogel and B. David, "A parallel processing approach to a multiple traveling salesman problem using evolutionary programming." L. Canter, Fullerton, CA,, 1990, pp. 318–326.

[135] O. Abdoun, J. Abouchabaka, and t. Tajani, "Analyzing the performance of mutation operators to solve the travelling salesman problem," *arXiv preprint arXiv:1203.3099*, 2012.

[136] P. Merz and B. Freisleben, "Genetic local search for the TSP: New results," in *Evolutionary Computation, 1997., IEEE International Conference on*. IEEE, 1997, pp. 159–164.

[137] B. F. Al-Dulaimi and H. A. Ali, "Enhanced traveling salesman problem solving by genetic algorithm technique (TSPGA)," *World Academy of Science, Engineering and Technology*, vol. 38, pp. 296–302, 2008.

[138] F. Liu and G. Zeng, "Study of genetic algorithm with reinforcement learning to solve the TSP," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6995–7001, 2009.

[139] L. Jiao and L. Wang, "A novel genetic algorithm based on immunity," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 30, no. 5, pp. 552–561, 2000.

[140] A. Gupta, P. Chandna, and P. Tandon, "Hybrid genetic algorithm for minimizing non productive machining time during 2.5D milling," *International Journal of Engineering, Science and Technology*, vol. 3, no. 1, 2011.

[141] A. Uğur, "Path planning on a cuboid using genetic algorithms," *Information Sciences*, vol. 178, no. 16, pp. 3275–3287, 2008.

[142] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: An autocatalytic optimizing process," *TR91-016, Politecnico di Milano*, 1991.

[143] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions*, vol. 26, no. 1, pp. 29–41, 1996.

[144] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997.

[145] G. S. Tewolde and W. Sheng, "Robot path integration in manufacturing processes: Genetic algorithm versus ant colony optimization," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 38, no. 2, pp. 278–287, 2008.

[146] M. Manfrin, M. Birattari, T. Stützle, and M. Dorigo, "Parallel ant colony optimization for the traveling salesman problem," in *Ant Colony Optimization and Swarm Intelligence*. Springer, 2006, pp. 224–234.

[147] C. F. Tsai, C. W. Tsai, and C. C. Tseng, "A new hybrid heuristic approach for solving large traveling salesman problem," *Information Sciences*, vol. 166, no. 1, pp. 67–81, 2004.

[148] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[149] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of statistical physics*, vol. 34, no. 5-6, pp. 975–986, 1984.

[150] D. Bookstaber, "Simulated annealing for traveling salesman problem," 1997.

[151] M. Malek, M. Guruswamy, M. Pandya, and H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 21, no. 1, pp. 59–84, 1989.

[152] Z. Car, T. Mikac, and I. Veža, "Utilization of GA for optimization of tool path on a 2D surface," in *Proceedings of 6th International Workshop on Emergent Synthesis IWES*, vol. 6, 2006, pp. 231–236.

[153] K. Weinert, J. Mehnen, and M. Stautner, "The application of multiobjective evolutionary algorithms to the generation of optimized tool paths for multi-axis die and mould making," in *Intelligent Computation in Manufacturing Engineering, 4th CIRP International Seminar on Intelligent Computation in Manufacturing Engineering, CIRP ICME*, vol. 4, 2004, pp. 406–412.

[154] J. Mehnen, R. Roy, P. Kersting, and T. Wagner, "ICSPEA: evolutionary five-axis milling path optimisation," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 2122–2128.

[155] P. Kersting and A. Zabel, "Optimizing NC-tool paths for simultaneous five-axis milling based on multi-population multi-objective evolutionary algorithms," *Advances in Engineering Software*, vol. 40, no. 6, pp. 452–463, 2009.

[156] L. Jiao, H. Wang, R. Shang, and F. Liu, "A co-evolutionary multi-objective optimization algorithm based on direction vectors," *Information Sciences*, 2012.

[157] D. J. Mundform, J. Schaffer, M.-J. Kim, D. Shaw, A. Thongteeraparp, and P. Supawan, "Number of replications required in monte carlo simulation studies: a synthesis of four studies," *Journal of Modern Applied Statistical Methods*, vol. 10, no. 1, p. 4, 2011.

[158] B. Iooss and P. Lemaître, "A review on global sensitivity analysis methods," in *Uncertainty Management in Simulation-Optimization of Complex Systems*. Springer, 2015, pp. 101–122.

[159] G. Reinelt, "Tspliba traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[160] R. L. Liu, C. R. Zhang, A. Nassehi, and S. Newman, "A step-nc programming system for prismatic parts," in *Materials Science Forum*, vol. 532. Trans Tech Publ, 2006, pp. 1108–1111.

[161] F. Feito, J. C. Torres, and A. Urena, "Orientation, simplicity, and inclusion test for planar polygons," *Computers & Graphics*, vol. 19, no. 4, pp. 595–600, 1995.

[162] G. Taylor, "Point in polygon test," *Survey Review*, vol. 32, no. 254, pp. 479–484, 1994.

[163] W. P. Essink, A. Nassehi, and S. T. Newman, "Toolpath generation for CNC milled parts using genetic algorithms," in *Enabling Manufacturing Competitiveness and Economic Sustainability*. Springer, 2014, pp. 189–193.

[164] J. Bresenham, "A linear algorithm for incremental digital display of circular arcs," *Communications of the ACM*, vol. 20, no. 2, pp. 100–106, 1977.

[165] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems journal*, vol. 4, no. 1, pp. 25–30, 1965.

[166] "Fishhead - step tools, inc." steptools.com/products/stepncmachine/samples/fishhead/, accessed: 2014-03-10.

[167] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[168] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 2. IEEE, 2000, pp. 1354–1360.

[169] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE transactions on evolutionary computation*, vol. 6, no. 6, pp. 580–593, 2002.

[170] D. Goswami, H. Karnick, P. Jain, and H. K. Maji, "Towards efficiently solving quantum traveling salesman problem," *arXiv preprint quant-ph/0411013*, 2004.

Chapter 9

Appendix

9.1 STEP-NC Programme of Test Parts

9.1.1 Fishhead Test Part

```
ISO-10303-21
HEADER;
FILE_DESCRIPTION(
('Fishhead Test Part'),
'2;1');
FILE_NAME(
'TEST.STP',
('Wesley Essink','Xianzhi Zhang','Aydin Nassehi'),
('University of Bath'),
' ',
'STEPMAN',
' ');
FILE_SCHEMA(('COMBINED_Schema'));
ENDSEC;
DATA;
#1=PROJECT('"RECOGNISED ISO 14649 PART 21  FILE FROM G&M CODES"',#2,(#3),$,$,$);
#2=WORKPLAN('"MAIN WORKPLAN"',(#4,#5,#6,#7,#8,#9,#10,#11,#12,#13),$,#14,$);
#3=WORKPIECE('"WORKPIECE135.0X185.0X40.0"',$,$,$,$,#24,());
#4=MACHINING_WORKINGSTEP('"WS "PLANAR_FACE1""',#15,#16,#17,$);
#5=MACHINING_WORKINGSTEP('"WS "PLANAR_FACE2""',#15,#41,#42,$);
#6=MACHINING_WORKINGSTEP('"WS "PLANAR_FACE3""',#15,#120,#121,$);
#7=MACHINING_WORKINGSTEP('"WS "POCKET1""',#15,#216,#217,$);
#8=MACHINING_WORKINGSTEP('"WS "POCKET2""',#15,#289,#290,$);
#9=MACHINING_WORKINGSTEP('"WS "POCKET3""',#15,#358,#359,$);
#10=MACHINING_WORKINGSTEP('"WS "POCKET4""',#15,#431,#432,$);
#11=MACHINING_WORKINGSTEP('"WS "POCKET5""',#15,#500,#501,$);
#12=MACHINING_WORKINGSTEP('"WS "POCKET6""',#15,#597,#598,$);
#13=MACHINING_WORKINGSTEP('"WS "PLANAR_FACE4""',#15,#618,#619,$);
#14=SETUP('"SETUP"',#792,#15,(#793));
#15=ELEMENTARY_SURFACE('"SECURITY PLANE"',#18);
#16=PLANAR_FACE('"PLANAR_FACE1"',#3,(#17),#22,#23,$,$,$,());
#17=PLANE_MILLING($,$,'"PLANAR_FACE1"',$,$,#29,#30,#31,$,$,$,$,$,$);
#18=AXIS2_PLACEMENT_3D('"SECURITY PLANE PLACEMENT"',#19,#20,#21);
#19=CARTESIAN_POINT('"SECURITY PLANE: LOCATION"',(0.0,0.0,10.0,0.0,0.0));
#20=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#21=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#22=AXIS2_PLACEMENT_3D('"PLANAR_FACE1 PLACEMENT"',#34,#35,#36);
#23=ELEMENTARY_SURFACE('"PLANAR_FACE1 DEPTH PLANE"',#37);
#24=BLOCK('"WORKPIECE BLOCK"',#25,135.0,185.0,-40.0);
#25=AXIS2_PLACEMENT_3D('""WORKPIECE BLOCK" PLACEMENT"',#26,#27,#28);
#26=CARTESIAN_POINT('""WORKPIECE BLOCK": LOCATION"',(0.0,0.0,0.0));
#27=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#28=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#29=MILLING_CUTTING_TOOL('"T4"',#32,(),$,$,$);
#30=MILLING_TECHNOLOGY(0.0041666666666666667,.TCP.,$,33.15,$,$,$,$,$);
#31=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#32=FACEMILL(#33,$,$,$,$);
#33=MILLING_TOOL_DIMENSION(40.0,$,$,$,0.0,$,$);
#34=CARTESIAN_POINT('"PLANAR_FACE1"',(175.0,8.125,0.0));
#35=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#36=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#37=AXIS2_PLACEMENT_3D('"PLANAR_FACE1 DEPTH"',#38,#39,#40);
#38=CARTESIAN_POINT('"PLANAR_FACE1 DEPTH"',(0.0,0.0,-2.0));
#39=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#40=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#41=PLANAR_FACE('"PLANAR_FACE2"',#3,(#42),#43,#44,$,$,$,(#45));
```

```
#42=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE2'',$,$,#46,#47,#48,$,$,$,$,$,$,$,$);
#43=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 PLACEMENT'',#51,#52,#53);
#44=ELEMENTARY_SURFACE('' PLANAR_FACE2 DEPTH PLANE'',#54);
#45=BOSS('' PLANAR_FACE2 BOSS'',#3,(),#58,#44,#59,$);
#46=MILLING_CUTTING_TOOL('' T3'',#49,(),$,$,$);
#47=MILLING_TECHNOLOGY(0.013266666666666666,.TCP.,$,132.63333333333333,$,$,$,$,$);
#48=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#49=ENDMILL(#50,$,$,$,$);
#50=MILLING_TOOL_DIMENSION(20.0,$,$,$,0.0,$,$);
#51=CARTESIAN_POINT('' PLANAR_FACE2'',(144.94,112.042,-4.0));
#52=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#53=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#54=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 DEPTH'',#55,#56,#57);
#55=CARTESIAN_POINT('' PLANAR_FACE2 DEPTH'',(-65.0,0.0,-12.0));
#56=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#57=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#58=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 PLACEMENT'',#60,#61,#62);
#59=GENERAL_CLOSED_PROFILE($,#63);
#60=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE2 BOSS"'',(-115.166,20.014000000000001,0.0));
#61=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#62=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#63=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE2'',(#64,#65,#66,#67,#68),.F.);
#64=COMPOSITE_CURVE_SEGMENT($,.T.,#71);
#65=COMPOSITE_CURVE_SEGMENT($,.T.,#75);
#66=COMPOSITE_CURVE_SEGMENT($,.T.,#83);
#67=COMPOSITE_CURVE_SEGMENT($,.T.,#88);
#68=COMPOSITE_CURVE_SEGMENT($,.T.,#96);
#71=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#72,#73,#74));
#72=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.0,0.0,0.0));
#73=CARTESIAN_POINT('' POLYLINE POINT 2'',(-2.7720000000000056,-25.256000000000014,0.0));
#74=CARTESIAN_POINT('' POLYLINE POINT 2'',(-4.754000000000005,-67.60600000000001,-7.0));
#75=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE2'',#76,(#77),(#78),$,$);
#76=CIRCLE('' CIRCLE'',#79,5.0);
#77=CARTESIAN_POINT('' TRIM POINT 1'',(-4.754000000000005,-67.60600000000001,0.0));
#78=CARTESIAN_POINT('' TRIM POINT 2'',(0.2459999999999951,-72.60600000000001,0.0));
#79=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#80,#81,#82);
#80=CARTESIAN_POINT('' CIRCLE CENTER'',(0.2459999999999951,-67.60600000000001,0.0));
#81=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#82=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#83=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#84,#85,#86,#87));
#84=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.2459999999999951,-72.60600000000001,0.0));
#85=CARTESIAN_POINT('' POLYLINE POINT 2'',(22.676000000000002,-72.60600000000001,0.0));
#86=CARTESIAN_POINT('' POLYLINE POINT 2'',(22.676000000000002,-74.50600000000001,-12.0));
#87=CARTESIAN_POINT('' POLYLINE POINT 3'',(-0.5970000000000084,-74.55200000000002,-7.0));
#88=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE2'',#89,(#90),(#91),$,$);
#89=CIRCLE('' CIRCLE'',#92,5.0);
#90=CARTESIAN_POINT('' TRIM POINT 1'',(-0.5970000000000084,-74.55200000000002,0.0));
#91=CARTESIAN_POINT('' TRIM POINT 2'',(-5.373999999999995,-79.54700000000001,0.0));
#92=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#93,#94,#95);
#93=CARTESIAN_POINT('' CIRCLE CENTER'',(-0.3739999999999952,-79.54600000000002,0.0));
#94=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#95=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#96=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#97,#98,#99,#100,#101,#102,#103));
#97=CARTESIAN_POINT('' POLYLINE POINT 1'',(-5.373999999999995,-79.54700000000001,0.0));
#98=CARTESIAN_POINT('' POLYLINE POINT 2'',(-5.373999999999995,-84.608,0.0));
#99=CARTESIAN_POINT('' POLYLINE POINT 2'',(-7.813999999999993,-103.93,-12.0));
#100=CARTESIAN_POINT('' POLYLINE POINT 3'',(-14.774000000000001,-103.93,-7.0));
#101=CARTESIAN_POINT('' POLYLINE POINT 4'',(-14.774000000000001,-49.902000000000015,-12.0));
#102=CARTESIAN_POINT('' POLYLINE POINT 5'',(-6.025999999999996,0.0,-12.0));
#103=CARTESIAN_POINT('' POLYLINE POINT 6'',(0.0,0.0,-7.0));
#120=PLANAR_FACE('' PLANAR_FACE3'',#3,(#121),#122,#123,$,$,$,(#124));
#121=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE3'',$,$,#46,#125,#126,$,$,$,$,$,$,$,$);
#122=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 PLACEMENT'',#127,#128,#129);
#123=ELEMENTARY_SURFACE('' PLANAR_FACE3 DEPTH PLANE'',#130);
#124=BOSS('' PLANAR_FACE3 BOSS'',#3,(),#134,#123,#135,$);
#125=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,127.31666666666666,$,$,$,$,$);
#126=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#127=CARTESIAN_POINT('' PLANAR_FACE3'',(1.694,194.94,-14.0));
#128=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#129=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#130=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 DEPTH'',#131,#132,#133);
#131=CARTESIAN_POINT('' PLANAR_FACE3 DEPTH'',(-11.634,-21.989000000000004,-23.5));
#132=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#133=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#134=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 PLACEMENT'',#136,#137,#138);
#135=GENERAL_CLOSED_PROFILE($,#139);
#136=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE3 BOSS"'',(99.206,-27.439999999999998,0.0));
#137=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#138=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#139=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE3'',(#140,#141,#142,#143,#144,#145,#146,#147,#148),.F.);
#140=COMPOSITE_CURVE_SEGMENT($,.T.,#149);
#141=COMPOSITE_CURVE_SEGMENT($,.T.,#163);
#142=COMPOSITE_CURVE_SEGMENT($,.T.,#171);
#143=COMPOSITE_CURVE_SEGMENT($,.T.,#174);
#144=COMPOSITE_CURVE_SEGMENT($,.T.,#182);
#145=COMPOSITE_CURVE_SEGMENT($,.T.,#192);
#146=COMPOSITE_CURVE_SEGMENT($,.T.,#200);
#147=COMPOSITE_CURVE_SEGMENT($,.T.,#203);
```

```
#148=COMPOSITE_CURVE_SEGMENT($,.T.,#211);
#149=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE3"',(#150,#151,#152,#153,#154,#155,#156,#157,#158,#159,
                                                      #160,#161,#162));
#150=CARTESIAN_POINT('"POLYLINE POINT 1"',(0.0,0.0,0.0));
#151=CARTESIAN_POINT('"POLYLINE POINT 2"',(3.100999999999999,-6.0020000000000095,0.0));
#152=CARTESIAN_POINT('"POLYLINE POINT 2"',(9.549999999999997,-36.75,-15.0));
#153=CARTESIAN_POINT('"POLYLINE POINT 3"',(18.099999999999994,-86.85,-20.0));
#154=CARTESIAN_POINT('"POLYLINE POINT 4"',(18.099999999999994,-152.5,-15.0));
#155=CARTESIAN_POINT('"POLYLINE POINT 5"',(-85.9,-152.5,-15.0));
#156=CARTESIAN_POINT('"POLYLINE POINT 6"',(-85.9,-85.346,-20.0));
#157=CARTESIAN_POINT('"POLYLINE POINT 7"',(-75.9,-28.300000000000001,-15.0));
#158=CARTESIAN_POINT('"POLYLINE POINT 8"',(-69.4,0.0,-20.0));
#159=CARTESIAN_POINT('"POLYLINE POINT 9"',(-66.4,0.0,-15.0));
#160=CARTESIAN_POINT('"POLYLINE POINT 10"',(-71.13300000000001,-35.50200000000001,-15.0));
#161=CARTESIAN_POINT('"POLYLINE POINT 11"',(-73.898,-60.7,-15.0));
#162=CARTESIAN_POINT('"POLYLINE POINT 12"',(-75.88,-103.05000000000001,-15.0));
#163=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3"',#164,(#165),(#166),$,$);
#164=CIRCLE('"CIRCLE"',#167,5.0);
#165=CARTESIAN_POINT('"TRIM POINT 1"',(-75.88,-103.05000000000001,0.0));
#166=CARTESIAN_POINT('"TRIM POINT 2"',(-70.88,-108.05000000000001,0.0));
#167=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#168,#169,#170);
#168=CARTESIAN_POINT('"CIRCLE CENTER"',(-70.88,-103.05000000000001,0.0));
#169=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#170=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#171=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE3"',(#172,#173));
#172=CARTESIAN_POINT('"POLYLINE POINT 1"',(-70.88,-108.05000000000001,0.0));
#173=CARTESIAN_POINT('"POLYLINE POINT 2"',(-41.900000000000006,-108.05000000000001,0.0));
#174=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3"',#175,(#176),(#177),$,$);
#175=CIRCLE('"CIRCLE"',#178,5.0);
#176=CARTESIAN_POINT('"TRIM POINT 1"',(-41.900000000000006,-108.05000000000001,0.0));
#177=CARTESIAN_POINT('"TRIM POINT 2"',(-36.900000000000006,-103.05000000000001,0.0));
#178=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#179,#180,#181);
#179=CARTESIAN_POINT('"CIRCLE CENTER"',(-41.900000000000006,-103.05000000000001,0.0));
#180=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#181=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#182=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE3"',(#183,#184,#185,#186,#187,#188,#189,#190,#191));
#183=CARTESIAN_POINT('"POLYLINE POINT 1"',(-36.900000000000006,-103.05000000000001,0.0));
#184=CARTESIAN_POINT('"POLYLINE POINT 2"',(-36.90200000000001,-65.567,0.0));
#185=CARTESIAN_POINT('"POLYLINE POINT 2"',(-36.91100000000001,-65.435,2.0));
#186=CARTESIAN_POINT('"POLYLINE POINT 3"',(-36.90800000000001,-65.416,-20.0));
#187=CARTESIAN_POINT('"POLYLINE POINT 4"',(-36.932,-65.132,-15.0));
#188=CARTESIAN_POINT('"POLYLINE POINT 5"',(-36.94200000000001,-60.68899999999999,-15.0));
#189=CARTESIAN_POINT('"POLYLINE POINT 6"',(-33.971000000000004,-60.68899999999999,2.0));
#190=CARTESIAN_POINT('"POLYLINE POINT 7"',(-34.94600000000001,-73.05,-15.0));
#191=CARTESIAN_POINT('"POLYLINE POINT 8"',(-34.95,-103.05000000000001,-15.0));
#192=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3"',#193,(#194),(#195),$,$);
#193=CIRCLE('"CIRCLE"',#196,5.0);
#194=CARTESIAN_POINT('"TRIM POINT 1"',(-34.95,-103.05000000000001,0.0));
#195=CARTESIAN_POINT('"TRIM POINT 2"',(-29.950000000000003,-108.05000000000001,0.0));
#196=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#197,#198,#199);
#197=CARTESIAN_POINT('"CIRCLE CENTER"',(-29.950000000000003,-103.05000000000001,0.0));
#198=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#199=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#200=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE3"',(#201,#202));
#201=CARTESIAN_POINT('"POLYLINE POINT 1"',(-29.950000000000003,-108.05000000000001,0.0));
#202=CARTESIAN_POINT('"POLYLINE POINT 2"',(6.941999999999993,-108.04599999999999,0.0));
#203=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3"',#204,(#205),(#206),$,$);
#204=CIRCLE('"CIRCLE"',#207,5.0);
#205=CARTESIAN_POINT('"TRIM POINT 1"',(6.941999999999993,-108.04599999999999,0.0));
#206=CARTESIAN_POINT('"TRIM POINT 2"',(11.713999999999999,-102.63,0.0));
#207=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#208,#209,#210);
#208=CARTESIAN_POINT('"CIRCLE CENTER"',(6.72999999999999,-103.05000000000001,0.0));
#209=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#210=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#211=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE3"',(#212,#213,#214,#215));
#212=CARTESIAN_POINT('"POLYLINE POINT 1"',(11.713999999999999,-102.63,0.0));
#213=CARTESIAN_POINT('"POLYLINE POINT 2"',(6.5479999999999988,-41.982,0.0));
#214=CARTESIAN_POINT('"POLYLINE POINT 2"',(-1.3010000000000002,0.0,-20.0));
#215=CARTESIAN_POINT('"POLYLINE POINT 3"',(0.0,0.0,-15.0));
#216=CLOSED_POCKET('"POCKET1"',#3,(#217),#218,#219,(),$,#220,#221,$,#222);
#217=BOTTOM_AND_SIDE_MILLING($,$,'"POCKET1"',$,$,#46,#223,#224,$,$,$,$,$,$,$);
#218=AXIS2_PLACEMENT_3D('"POCKET1 PLACEMENT"',#225,#226,#227);
#219=ELEMENTARY_SURFACE('"POCKET1 DEPTH PLANE"',#228);
#220=PLANAR_POCKET_BOTTOM_CONDITION();
#221=TOLERANCED_LENGTH_MEASURE(5.0,$);
#222=GENERAL_CLOSED_PROFILE($,#232);
#223=MILLING_TECHNOLOGY(0.008483333333333334,.TCP.,$,106.1,$,$,$,$,$);
#224=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#225=CARTESIAN_POINT('"POCKET1"',(29.924,47.056,-14.0));
#226=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#227=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#228=AXIS2_PLACEMENT_3D('"POCKET1 DEPTH"',#229,#230,#231);
#229=CARTESIAN_POINT('"POCKET1 DEPTH"',(-0.024000000000000091,0.4620000000000033,-17.0));
#230=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#231=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#232=COMPOSITE_CURVE('"BOUNDARY: POCKET1"',(#233,#234,#235,#236,#237,#238,#239,#240,#241),.F.);
#233=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#242);
#234=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#250);
```

```
#235=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#253);
#236=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#256);
#237=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#264);
#238=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#267);
#239=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#275);
#240=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#278);
#241=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#286);
#242=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#243,(#244),(#245),.T.,.CARTESIAN.);
#243=CIRCLE('"CIRCLE"',#246,5.0);
#244=CARTESIAN_POINT('"TRIM POINT 1"',(-0.7469999999999999,10.448,0.0));
#245=CARTESIAN_POINT('"TRIM POINT 2"',(-5.524000000000001,5.453000000000003,0.0));
#246=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#247,#248,#249);
#247=CARTESIAN_POINT('"CIRCLE CENTER"',(-0.5240000000000009,5.454000000000001,0.0));
#248=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#249=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#250=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#251,#252));
#251=CARTESIAN_POINT('"POLYLINE POINT 1"',(-5.524000000000001,5.453000000000003,0.0));
#252=CARTESIAN_POINT('"POLYLINE POINT 2"',(-5.524000000000001,0.392000000000003,0.0));
#253=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#254,#255));
#254=CARTESIAN_POINT('"POLYLINE POINT 1"',(-5.524000000000001,0.392000000000003,0.0));
#255=CARTESIAN_POINT('"POLYLINE POINT 2"',(-7.983999999999998,-19.119999999999997,0.0));
#256=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#257,(#258),(#259),.T.,.CARTESIAN.);
#257=CIRCLE('"CIRCLE"',#260,5.0);
#258=CARTESIAN_POINT('"TRIM POINT 1"',(-7.983999999999998,-19.119999999999997,0.0));
#259=CARTESIAN_POINT('"TRIM POINT 2"',(-3.003,-24.555999999999997,0.0));
#260=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#261,#262,#263);
#261=CARTESIAN_POINT('"CIRCLE CENTER"',(-3.003999999999998,-19.555999999999997,0.0));
#262=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#263=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#264=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#265,#266));
#265=CARTESIAN_POINT('"POLYLINE POINT 1"',(-3.003,-24.555999999999997,0.0));
#266=CARTESIAN_POINT('"POLYLINE POINT 2"',(76.876,-24.555999999999997,0.0));
#267=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#268,(#269),(#270),.T.,.CARTESIAN.);
#268=CIRCLE('"CIRCLE"',#271,5.0);
#269=CARTESIAN_POINT('"TRIM POINT 1"',(76.876,-24.555999999999997,0.0));
#270=CARTESIAN_POINT('"TRIM POINT 2"',(81.876,-19.555999999999997,0.0));
#271=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#272,#273,#274);
#272=CARTESIAN_POINT('"CIRCLE CENTER"',(76.876,-19.555999999999997,0.0));
#273=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#274=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#275=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#276,#277));
#276=CARTESIAN_POINT('"POLYLINE POINT 1"',(81.876,-19.555999999999997,0.0));
#277=CARTESIAN_POINT('"POLYLINE POINT 2"',(81.87100000000001,5.631,0.0));
#278=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#279,(#280),(#281),.T.,.CARTESIAN.);
#279=CIRCLE('"CIRCLE"',#282,5.0);
#280=CARTESIAN_POINT('"TRIM POINT 1"',(81.87100000000001,5.631,0.0));
#281=CARTESIAN_POINT('"TRIM POINT 2"',(77.096,10.410000000000004,0.0));
#282=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#283,#284,#285);
#283=CARTESIAN_POINT('"CIRCLE CENTER"',(76.876,5.4140000000000015,0.0));
#284=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#285=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#286=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#287,#288));
#287=CARTESIAN_POINT('"POLYLINE POINT 1"',(77.096,10.410000000000004,0.0));
#288=CARTESIAN_POINT('"POLYLINE POINT 2"',(-0.7469999999999999,10.448,0.0));
#289=CLOSED_POCKET('"POCKET2"',#3,(#290),#291,#292,(),$,#293,#294,$,#295);
#290=BOTTOM_AND_SIDE_MILLING($,$,'"POCKET2"',$,$,#46,#296,#297,$,$,$,$,$,$,$,$);
#291=AXIS2_PLACEMENT_3D('"POCKET2 PLACEMENT"',#298,#299,#300);
#292=ELEMENTARY_SURFACE('"POCKET2 DEPTH PLANE"',#301);
#293=PLANAR_POCKET_BOTTOM_CONDITION();
#294=TOLERANCED_LENGTH_MEASURE(5.0,$);
#295=GENERAL_CLOSED_PROFILE($,#305);
#296=MILLING_TECHNOLOGY(0.008483333333333334,.TCP.,$,106.63333333333334,$,$,$,$,$);
#297=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#298=CARTESIAN_POINT('"POCKET2"',(101.797,47.475,-19.0));
#299=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#300=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#301=AXIS2_PLACEMENT_3D('"POCKET2 DEPTH"',#302,#303,#304);
#302=CARTESIAN_POINT('"POCKET2 DEPTH"',(4.503,-14.975000000000001,-24.5));
#303=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#304=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#305=COMPOSITE_CURVE('"BOUNDARY: POCKET2"',(#306,#307,#308,#309,#310,#311,#312,#313),.F.);
#306=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#314);
#307=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#322);
#308=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#325);
#309=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#333);
#310=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#336);
#311=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#344);
#312=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#347);
#313=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#355);
#314=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET2"',#315,(#316),(#317),.T.,.CARTESIAN.);
#315=CIRCLE('"CIRCLE"',#318,5.0);
#316=CARTESIAN_POINT('"TRIM POINT 1"',(-31.063000000000002,10.015999999999998,0.0));
#317=CARTESIAN_POINT('"TRIM POINT 2"',(-35.842,5.241,0.0));
#318=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#319,#320,#321);
#319=CARTESIAN_POINT('"CIRCLE CENTER"',(-30.846999999999994,5.024999999999999,0.0));
#320=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#321=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#322=POLYLINE('"POLYLINE FOR CONTOUR: POCKET2"',(#323,#324));
```

```
#323=CARTESIAN_POINT('' POLYLINE POINT 1'',(-35.842,5.241,0.0));
#324=CARTESIAN_POINT('' POLYLINE POINT 2'',(-35.846999999999994,-19.975,0.0));
#325=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET2'',#326,(#327),(#328),.T.,.CARTESIAN.);
#326=CIRCLE('' CIRCLE'',#329,5.0);
#327=CARTESIAN_POINT('' TRIM POINT 1'',(-35.846999999999994,-19.975,0.0));
#328=CARTESIAN_POINT('' TRIM POINT 2'',(-30.846999999999994,-24.975,0.0));
#329=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#330,#331,#332);
#330=CARTESIAN_POINT('' CIRCLE CENTER'',(-30.846999999999994,-19.975,0.0));
#331=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#332=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#333=POLYLINE('' POLYLINE FOR CONTOUR: POCKET2'',(#334,#335));
#334=CARTESIAN_POINT('' POLYLINE POINT 1'',(-30.846999999999994,-24.975,0.0));
#335=CARTESIAN_POINT('' POLYLINE POINT 2'',(5.003,-24.975,0.0));
#336=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET2'',#337,(#338),(#339),.T.,.CARTESIAN.);
#337=CIRCLE('' CIRCLE'',#340,5.0);
#338=CARTESIAN_POINT('' TRIM POINT 1'',(5.003,-24.975,0.0));
#339=CARTESIAN_POINT('' TRIM POINT 2'',(10.003,-19.975,0.0));
#340=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#341,#342,#343);
#341=CARTESIAN_POINT('' CIRCLE CENTER'',(5.003,-19.975,0.0));
#342=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#343=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#344=POLYLINE('' POLYLINE FOR CONTOUR: POCKET2'',(#345,#346));
#345=CARTESIAN_POINT('' POLYLINE POINT 1'',(10.003,-19.975,0.0));
#346=CARTESIAN_POINT('' POLYLINE POINT 2'',(9.998000000000005,5.211999999999996,0.0));
#347=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET2'',#348,(#349),(#350),.T.,.CARTESIAN.);
#348=CIRCLE('' CIRCLE'',#351,5.0);
#349=CARTESIAN_POINT('' TRIM POINT 1'',(9.998000000000005,5.211999999999996,0.0));
#350=CARTESIAN_POINT('' TRIM POINT 2'',(5.222999999999999,9.991,0.0));
#351=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#352,#353,#354);
#352=CARTESIAN_POINT('' CIRCLE CENTER'',(5.003,4.994999999999997,0.0));
#353=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#354=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#355=POLYLINE('' POLYLINE FOR CONTOUR: POCKET2'',(#356,#357));
#356=CARTESIAN_POINT('' POLYLINE POINT 1'',(5.222999999999999,9.991,0.0));
#357=CARTESIAN_POINT('' POLYLINE POINT 2'',(-31.063000000000002,10.015999999999998,0.0));
#358=CLOSED_POCKET('' POCKET3'',#3,(#359),#360,#361,(),$,#362,#363,$,#364);
#359=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET3'',$,$,#46,#365,#366,$,$,$,$,$,$,$);
#360=AXIS2_PLACEMENT_3D('' POCKET3 PLACEMENT'',#367,#368,#369);
#361=ELEMENTARY_SURFACE('' POCKET3 DEPTH PLANE'',#370);
#362=PLANAR_POCKET_BOTTOM_CONDITION();
#363=TOLERANCED_LENGTH_MEASURE(5.0,$);
#364=GENERAL_CLOSED_PROFILE($,#374);
#365=MILLING_TECHNOLOGY(0.006666666666666667,.TCP.,$,106.1,$,$,$,$,$);
#366=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#367=CARTESIAN_POINT('' POCKET3'',(32.592,32.5,-19.0));
#368=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#369=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#370=AXIS2_PLACEMENT_3D('' POCKET3 DEPTH'',#371,#372,#373);
#371=CARTESIAN_POINT('' POCKET3 DEPTH'',(-2.692,19.509999999999998,-24.5));
#372=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#373=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#374=COMPOSITE_CURVE('' BOUNDARY: POCKET3'',(#375,#376,#377,#378,#379,#380,#381,#382,#383),.F.);
#375=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#384);
#376=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#392);
#377=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#395);
#378=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#398);
#379=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#406);
#380=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#409);
#381=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#417);
#382=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#420);
#383=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#428);
#384=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#385,(#386),(#387),.T.,.CARTESIAN.);
#385=CIRCLE('' CIRCLE'',#388,5.0);
#386=CARTESIAN_POINT('' TRIM POINT 1'',(-3.414999999999999,25.003999999999998,0.0));
#387=CARTESIAN_POINT('' TRIM POINT 2'',(-8.192,20.009,0.0));
#388=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#389,#390,#391);
#389=CARTESIAN_POINT('' CIRCLE CENTER'',(-3.192,20.009999999999998,0.0));
#390=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#391=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#392=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#393,#394));
#393=CARTESIAN_POINT('' POLYLINE POINT 1'',(-8.192,20.009,0.0));
#394=CARTESIAN_POINT('' POLYLINE POINT 2'',(-8.192,14.948,0.0));
#395=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#396,#397));
#396=CARTESIAN_POINT('' POLYLINE POINT 1'',(-8.192,14.948,0.0));
#397=CARTESIAN_POINT('' POLYLINE POINT 2'',(-10.651999999999997,-4.564,0.0));
#398=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#399,(#400),(#401),.T.,.CARTESIAN.);
#399=CIRCLE('' CIRCLE'',#402,5.0);
#400=CARTESIAN_POINT('' TRIM POINT 1'',(-10.651999999999997,-4.564,0.0));
#401=CARTESIAN_POINT('' TRIM POINT 2'',(-5.670999999999999,-10.0,0.0));
#402=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#403,#404,#405);
#403=CARTESIAN_POINT('' CIRCLE CENTER'',(-5.671999999999997,-5.0,0.0));
#404=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#405=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#406=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#407,#408));
#407=CARTESIAN_POINT('' POLYLINE POINT 1'',(-5.670999999999999,-10.0,0.0));
#408=CARTESIAN_POINT('' POLYLINE POINT 2'',(26.408,-10.0,0.0));
#409=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#410,(#411),(#412),.T.,.CARTESIAN.);
#410=CIRCLE('' CIRCLE'',#413,5.0);
```

```
#411=CARTESIAN_POINT('' TRIM POINT 1'',(26.408,-10.0,0.0));
#412=CARTESIAN_POINT('' TRIM POINT 2'',(31.408,-5.0,0.0));
#413=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#414,#415,#416);
#414=CARTESIAN_POINT('' CIRCLE CENTER'',(26.408,-5.0,0.0));
#415=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#416=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#417=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#418,#419));
#418=CARTESIAN_POINT('' POLYLINE POINT 1'',(31.408,-5.0,0.0));
#419=CARTESIAN_POINT('' POLYLINE POINT 2'',(31.404000000000003,20.229,0.0));
#420=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#421,(#422),(#423),.T.,.CARTESIAN.);
#421=CIRCLE('' CIRCLE'',#424,5.0);
#422=CARTESIAN_POINT('' TRIM POINT 1'',(31.404000000000003,20.229,0.0));
#423=CARTESIAN_POINT('' TRIM POINT 2'',(26.429000000000002,25.022,0.0));
#424=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#425,#426,#427);
#425=CARTESIAN_POINT('' CIRCLE CENTER'',(26.408,20.020000000000003,0.0));
#426=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#427=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#428=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#429,#430));
#429=CARTESIAN_POINT('' POLYLINE POINT 1'',(26.429000000000002,25.022,0.0));
#430=CARTESIAN_POINT('' POLYLINE POINT 2'',(-3.414999999999999,25.003999999999998,0.0));
#431=CLOSED_POCKET('' POCKET4'',#3,(#432),#433,#434,(),$,#435,#436,$,#437);
#432=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET4'',$,$,#46,#438,#439,$,$,$,$,$,$,$);
#433=AXIS2_PLACEMENT_3D('' POCKET4 PLACEMENT'',#440,#441,#442);
#434=ELEMENTARY_SURFACE('' POCKET4 DEPTH PLANE'',#443);
#435=PLANAR_POCKET_BOTTOM_CONDITION();
#436=TOLERANCED_LENGTH_MEASURE(5.0,$);
#437=GENERAL_CLOSED_PROFILE($,#447);
#438=MILLING_TECHNOLOGY(0.0138,.TCP.,$,137.93333333333334,$,$,$,$,$);
#439=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,(),$,$,$,());
#440=CARTESIAN_POINT('' POCKET4'',(55.508,92.56,-25.0));
#441=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#442=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#443=AXIS2_PLACEMENT_3D('' POCKET4 DEPTH'',#444,#445,#446);
#444=CARTESIAN_POINT('' POCKET4 DEPTH'',(-1.5080000000000027,8.739999999999995,-25.0));
#445=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#446=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#447=COMPOSITE_CURVE('' BOUNDARY: POCKET4'',(#448,#449,#450,#451,#452,#453,#454,#455),.F.);
#448=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#456);
#449=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#459);
#450=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#467);
#451=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#470);
#452=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#478);
#453=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#481);
#454=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#489);
#455=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#492);
#456=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#457,#458));
#457=CARTESIAN_POINT('' POLYLINE POINT 1'',(3.4919999999999973,14.239999999999995,0.0));
#458=CARTESIAN_POINT('' POLYLINE POINT 2'',(-23.939000000000004,14.236000000000004,0.0));
#459=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#460,(#461),(#462),.T.,.CARTESIAN.);
#460=CIRCLE('' CIRCLE'',#463,5.0);
#461=CARTESIAN_POINT('' TRIM POINT 1'',(-23.939000000000004,14.236000000000004,0.0));
#462=CARTESIAN_POINT('' TRIM POINT 2'',(-28.732000000000003,9.474000000000004,0.0));
#463=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#464,#465,#466);
#464=CARTESIAN_POINT('' CIRCLE CENTER'',(-23.738000000000003,9.239999999999995,0.0));
#465=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#466=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#467=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#468,#469));
#468=CARTESIAN_POINT('' POLYLINE POINT 1'',(-28.732000000000003,9.474000000000004,0.0));
#469=CARTESIAN_POINT('' POLYLINE POINT 2'',(-30.488000000000003,-28.11,0.0));
#470=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#471,(#472),(#473),.T.,.CARTESIAN.);
#471=CIRCLE('' CIRCLE'',#474,5.0);
#472=CARTESIAN_POINT('' TRIM POINT 1'',(-30.488000000000003,-28.11,0.0));
#473=CARTESIAN_POINT('' TRIM POINT 2'',(-25.488000000000003,-33.11,0.0));
#474=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#475,#476,#477);
#475=CARTESIAN_POINT('' CIRCLE CENTER'',(-25.488000000000003,-28.11,0.0));
#476=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#477=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#478=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#479,#480));
#479=CARTESIAN_POINT('' POLYLINE POINT 1'',(-25.488000000000003,-33.11,0.0));
#480=CARTESIAN_POINT('' POLYLINE POINT 2'',(3.4919999999999973,-33.11,0.0));
#481=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#482,(#483),(#484),.T.,.CARTESIAN.);
#482=CIRCLE('' CIRCLE'',#485,5.0);
#483=CARTESIAN_POINT('' TRIM POINT 1'',(3.4919999999999973,-33.11,0.0));
#484=CARTESIAN_POINT('' TRIM POINT 2'',(8.491999999999997,-28.11,0.0));
#485=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#486,#487,#488);
#486=CARTESIAN_POINT('' CIRCLE CENTER'',(3.4919999999999973,-28.11,0.0));
#487=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#488=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#489=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#490,#491));
#490=CARTESIAN_POINT('' POLYLINE POINT 1'',(8.491999999999997,-28.11,0.0));
#491=CARTESIAN_POINT('' POLYLINE POINT 2'',(8.491999999999997,9.239999999999995,0.0));
#492=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#493,(#494),(#495),.T.,.CARTESIAN.);
#493=CIRCLE('' CIRCLE'',#496,5.0);
#494=CARTESIAN_POINT('' TRIM POINT 1'',(8.491999999999997,9.239999999999995,0.0));
#495=CARTESIAN_POINT('' TRIM POINT 2'',(3.4919999999999973,14.239999999999995,0.0));
#496=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#497,#498,#499);
#497=CARTESIAN_POINT('' CIRCLE CENTER'',(3.4919999999999973,9.239999999999995,0.0));
#498=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
```

```
#499=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#500=CLOSED_POCKET('' POCKET5'',#3,(#501),#502,#503,(#504),$,#505,#506,$,#507);
#501=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET5'',$,$,#46,#508,#509,$,$,$,$,$,$,$);
#502=AXIS2_PLACEMENT_3D('' POCKET5 PLACEMENT'',#510,#511,#512);
#503=ELEMENTARY_SURFACE('' POCKET5 DEPTH PLANE'',#513);
#504=BOSS('' POCKET5_BOSS'',#3,(),#517,#503,#518,$);
#505=PLANAR_POCKET_BOTTOM_CONDITION();
#506=TOLERANCED_LENGTH_MEASURE(5.0,$);
#507=GENERAL_CLOSED_PROFILE($,#527);
#508=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,107.16666666666667,$,$,$,$,$);
#509=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#510=CARTESIAN_POINT('' POCKET5'',(104.64,87.373,-25.5));
#511=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#512=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#513=AXIS2_PLACEMENT_3D('' POCKET5 DEPTH'',#514,#515,#516);
#514=CARTESIAN_POINT('' POCKET5 DEPTH'',(0.8529999999999944,-4.927000000000007,-25.5));
#515=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#516=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#517=AXIS2_PLACEMENT_3D('' POCKET5 PLACEMENT'',#519,#520,#521);
#518=CIRCULAR_CLOSED_PROFILE(#522,#523);
#519=CARTESIAN_POINT('' LOCATIONPOCKET5'',(-18.040000000000006,-3.3230000000000075,-12.0));
#520=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#521=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#522=AXIS2_PLACEMENT_3D('' CIRCULAR PROFILE LOCATION'',#524,#525,#526);
#523=TOLERANCED_LENGTH_MEASURE(8.5,$);
#524=CARTESIAN_POINT('' LOCATION POINT'',(0.0,0.0,0.0));
#525=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#526=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#527=COMPOSITE_CURVE('' BOUNDARY: POCKET5'',(#528,#529,#530,#531,#532,#533,#534,#535,#536,#537,#538),.F.);
#528=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#539);
#529=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#547);
#530=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#550);
#531=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#558);
#532=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#561);
#533=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#564);
#534=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#572);
#535=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#575);
#536=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#583);
#537=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#586);
#538=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#594);
#539=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#540,(#541),(#542),.T.,.CARTESIAN.);
#540=CIRCLE('' CIRCLE'',#543,5.0);
#541=CARTESIAN_POINT('' TRIM POINT 1'',(-32.947,18.429000000000002,0.0));
#542=CARTESIAN_POINT('' TRIM POINT 2'',(-37.995000000000005,13.939999999999998,0.0));
#543=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#544,#545,#546);
#544=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.019999999999996,13.426999999999992,0.0));
#545=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#546=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#547=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#548,#549));
#548=CARTESIAN_POINT('' POLYLINE POINT 1'',(-37.995000000000005,13.939999999999998,0.0));
#549=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.022000000000006,13.527000000000001,0.0));
#550=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#551,(#552),(#553),.T.,.CARTESIAN.);
#551=CIRCLE('' CIRCLE'',#554,5.0);
#552=CARTESIAN_POINT('' TRIM POINT 1'',(-38.022000000000006,13.527000000000001,0.0));
#553=CARTESIAN_POINT('' TRIM POINT 2'',(-38.092,13.096999999999994,0.0));
#554=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#555,#556,#557);
#555=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.129999999999995,12.50699999999999,0.0));
#556=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#557=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#558=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#559,#560));
#559=CARTESIAN_POINT('' POLYLINE POINT 1'',(-38.092,13.096999999999994,0.0));
#560=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.566,7.0719999999999885,0.0));
#561=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#562,#563));
#562=CARTESIAN_POINT('' POLYLINE POINT 1'',(-38.566,7.0719999999999885,0.0));
#563=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.57000000000001,-22.80300000000001,0.0));
#564=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#565,(#566),(#567),.T.,.CARTESIAN.);
#565=CIRCLE('' CIRCLE'',#568,5.0);
#566=CARTESIAN_POINT('' TRIM POINT 1'',(-38.57000000000001,-22.80300000000001,0.0));
#567=CARTESIAN_POINT('' TRIM POINT 2'',(-33.57000000000001,-27.803000000000004,0.0));
#568=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#569,#570,#571);
#569=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.57000000000001,-22.80300000000001,0.0));
#570=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#571=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#572=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#573,#574));
#573=CARTESIAN_POINT('' POLYLINE POINT 1'',(-33.57000000000001,-27.803000000000004,0.0));
#574=CARTESIAN_POINT('' POLYLINE POINT 2'',(3.0720000000000027,-27.799000000000007,0.0));
#575=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#576,(#577),(#578),.T.,.CARTESIAN.);
#576=CIRCLE('' CIRCLE'',#579,5.0);
#577=CARTESIAN_POINT('' TRIM POINT 1'',(3.0720000000000027,-27.799000000000007,0.0));
#578=CARTESIAN_POINT('' TRIM POINT 2'',(7.843999999999994,-22.38300000000001,0.0));
#579=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#580,#581,#582);
#580=CARTESIAN_POINT('' CIRCLE CENTER'',(2.8599999999999994,-22.80300000000001,0.0));
#581=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#582=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#583=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#584,#585));
#584=CARTESIAN_POINT('' POLYLINE POINT 1'',(7.843999999999994,-22.38300000000001,0.0));
#585=CARTESIAN_POINT('' POLYLINE POINT 2'',(4.810999999999993,13.549999999999997,0.0));
#586=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#587,(#588),(#589),.T.,.CARTESIAN.);
```

```
#587=CIRCLE('' CIRCLE'',#590,5.0);
#588=CARTESIAN_POINT('' TRIM POINT 1'',(4.810999999999993,13.549999999999997,0.0));
#589=CARTESIAN_POINT('' TRIM POINT 2'',(-0.07800000000000296,17.94099999999999,0.0));
#590=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#591,#592,#593);
#591=CARTESIAN_POINT('' CIRCLE CENTER'',(-0.15000000000000568,12.946999999999989,0.0));
#592=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#593=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#594=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#595,#596));
#595=CARTESIAN_POINT('' POLYLINE POINT 1'',(-0.07800000000000296,17.94099999999999,0.0));
#596=CARTESIAN_POINT('' POLYLINE POINT 2'',(-32.947,18.429000000000002,0.0));
#597=CLOSED_POCKET('' POCKET6'',#3,(#598),#599,#600,(),$,#601,#602,$,#603);
#598=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET6'',$,$,#46,#604,#605,$,$,$,$,$,$,$,$);
#599=AXIS2_PLACEMENT_3D('' POCKET6 PLACEMENT'',#606,#607,#608);
#600=ELEMENTARY_SURFACE('' POCKET6 DEPTH PLANE'',#609);
#601=PLANAR_POCKET_BOTTOM_CONDITION();
#602=TOLERANCED_LENGTH_MEASURE(5.0,$);
#603=CIRCULAR_CLOSED_PROFILE(#613,#614);
#604=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,95.5,$,$,$,$,$);
#605=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#606=CARTESIAN_POINT('' POCKET6'',(86.6,84.055,-25.5));
#607=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#608=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#609=AXIS2_PLACEMENT_3D('' POCKET6 DEPTH'',#610,#611,#612);
#610=CARTESIAN_POINT('' POCKET6 DEPTH'',(0.0,0.0,-27.0));
#611=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#612=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#613=AXIS2_PLACEMENT_3D('' CIRCULAR PROFILE LOCATION'',#615,#616,#617);
#614=TOLERANCED_LENGTH_MEASURE(14.0,$);
#615=CARTESIAN_POINT('' LOCATION POINT'',(0.0,0.0,0.0));
#616=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#617=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#618=PLANAR_FACE('' PLANAR_FACE4'',#3,(#619),#620,#621,$,$,$,(#622));
#619=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE4'',$,$,#46,#623,#624,$,$,$,$,$,$,$,$);
#620=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 PLACEMENT'',#625,#626,#627);
#621=ELEMENTARY_SURFACE('' PLANAR_FACE4 DEPTH PLANE'',#628);
#622=BOSS('' PLANAR_FACE4 BOSS'',#3,(),#632,#621,#633,$);
#623=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,108.75,$,$,$,$,$);
#624=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#625=CARTESIAN_POINT('' PLANAR_FACE4'',(1.694,194.94,-25.5));
#626=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#627=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#628=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 DEPTH'',#629,#630,#631);
#629=CARTESIAN_POINT('' PLANAR_FACE4 DEPTH'',(-11.634,-21.989000000000004,-27.0));
#630=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#631=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#632=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 PLACEMENT'',#634,#635,#636);
#633=GENERAL_CLOSED_PROFILE($,#637);
#634=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE4 BOSS"'',(91.31099999999999,-27.439999999999998,0.0));
#635=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#636=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#637=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE4'',(#638,#639,#640,#641,#642,#643,#644,#645,#646,#647,#648,
                                                  #649,#650,#651,#652,#653,#654,#655,#656,#657,#658),.F.);
#638=COMPOSITE_CURVE_SEGMENT($,.T.,#659);
#639=COMPOSITE_CURVE_SEGMENT($,.T.,#676);
#640=COMPOSITE_CURVE_SEGMENT($,.T.,#684);
#641=COMPOSITE_CURVE_SEGMENT($,.T.,#687);
#642=COMPOSITE_CURVE_SEGMENT($,.T.,#695);
#643=COMPOSITE_CURVE_SEGMENT($,.T.,#699);
#644=COMPOSITE_CURVE_SEGMENT($,.T.,#707);
#645=COMPOSITE_CURVE_SEGMENT($,.T.,#710);
#646=COMPOSITE_CURVE_SEGMENT($,.T.,#718);
#647=COMPOSITE_CURVE_SEGMENT($,.T.,#721);
#648=COMPOSITE_CURVE_SEGMENT($,.T.,#729);
#649=COMPOSITE_CURVE_SEGMENT($,.T.,#732);
#650=COMPOSITE_CURVE_SEGMENT($,.T.,#740);
#651=COMPOSITE_CURVE_SEGMENT($,.T.,#743);
#652=COMPOSITE_CURVE_SEGMENT($,.T.,#751);
#653=COMPOSITE_CURVE_SEGMENT($,.T.,#754);
#654=COMPOSITE_CURVE_SEGMENT($,.T.,#762);
#655=COMPOSITE_CURVE_SEGMENT($,.T.,#765);
#656=COMPOSITE_CURVE_SEGMENT($,.T.,#773);
#657=COMPOSITE_CURVE_SEGMENT($,.T.,#776);
#658=COMPOSITE_CURVE_SEGMENT($,.T.,#784);
#659=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#660,#661,#662,#663,#664,#665,#666,#667,#668,#669,
                                                       #670,#671,#672,#673,#674,#675));
#660=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.0,0.0,0.0));
#661=CARTESIAN_POINT('' POLYLINE POINT 2'',(7.895000000000001,0.0,0.0));
#662=CARTESIAN_POINT('' POLYLINE POINT 2'',(10.996000000000001,-6.002000000000095,13.5));
#663=CARTESIAN_POINT('' POLYLINE POINT 3'',(17.445000000000007,-36.75,-13.5));
#664=CARTESIAN_POINT('' POLYLINE POINT 4'',(25.995000000000005,-86.85,13.5));
#665=CARTESIAN_POINT('' POLYLINE POINT 5'',(25.995000000000005,-152.5,-13.5));
#666=CARTESIAN_POINT('' POLYLINE POINT 6'',(-78.005,-152.5,-13.5));
#667=CARTESIAN_POINT('' POLYLINE POINT 7'',(-78.005,-85.346,13.5));
#668=CARTESIAN_POINT('' POLYLINE POINT 8'',(-68.005,-28.300000000000001,-13.5));
#669=CARTESIAN_POINT('' POLYLINE POINT 9'',(-61.504999999999995,0.0,13.5));
#670=CARTESIAN_POINT('' POLYLINE POINT 10'',(-58.504999999999995,0.0,-13.5));
#671=CARTESIAN_POINT('' POLYLINE POINT 11'',(-54.504999999999995,-0.25,13.5));
#672=CARTESIAN_POINT('' POLYLINE POINT 12'',(-52.629999999999995,-1.0999999999999943,13.5));
```

```
#673=CARTESIAN_POINT('"POLYLINE POINT 13"',(-51.2549999999999995,-2.49899999999999952,-13.5));
#674=CARTESIAN_POINT('"POLYLINE POINT 14"',(-50.404999999999994,-4.349999999999994,-13.5));
#675=CARTESIAN_POINT('"POLYLINE POINT 15"',(-41.55799999999999,-37.43700000000001,-13.5));
#676=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#677,(#678),(#679),$,$);
#677=CIRCLE('"CIRCLE"',#680,5.0);
#678=CARTESIAN_POINT('"TRIM POINT 1"',(-41.55799999999999,-37.43700000000001,0.0));
#679=CARTESIAN_POINT('"TRIM POINT 2"',(-41.340999999999994,-38.01599999999999,0.0));
#680=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#681,#682,#683);
#681=CARTESIAN_POINT('"CIRCLE CENTER"',(-36.775,-35.97,0.0));
#682=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#683=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#684=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#685,#686));
#685=CARTESIAN_POINT('"POLYLINE POINT 1"',(-41.340999999999994,-38.01599999999999,0.0));
#686=CARTESIAN_POINT('"POLYLINE POINT 2"',(-40.474,-39.943,0.0));
#687=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#688,(#689),(#690),$,$);
#688=CIRCLE('"CIRCLE"',#691,5.0);
#689=CARTESIAN_POINT('"TRIM POINT 1"',(-40.474,-39.943,0.0));
#690=CARTESIAN_POINT('"TRIM POINT 2"',(-39.751,-41.031000000000006,0.0));
#691=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#692,#693,#694);
#692=CARTESIAN_POINT('"CIRCLE CENTER"',(-35.98499999999999,-37.74000000000001,0.0));
#693=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#694=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#695=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#696,#697,#698));
#696=CARTESIAN_POINT('"POLYLINE POINT 1"',(-39.751,-41.031000000000006,0.0));
#697=CARTESIAN_POINT('"POLYLINE POINT 2"',(-38.69199999999999,-42.163,0.0));
#698=CARTESIAN_POINT('"POLYLINE POINT 2"',(-38.19499999999999,-42.656000000000006,-13.5));
#699=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#700,(#701),(#702),$,$);
#700=CIRCLE('"CIRCLE"',#703,5.0);
#701=CARTESIAN_POINT('"TRIM POINT 1"',(-38.19499999999999,-42.656000000000006,0.0));
#702=CARTESIAN_POINT('"TRIM POINT 2"',(-37.135,-43.419,0.0));
#703=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#704,#705,#706);
#704=CARTESIAN_POINT('"CIRCLE CENTER"',(-34.76499999999999,-39.00999999999999,0.0));
#705=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#706=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#707=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#708,#709));
#708=CARTESIAN_POINT('"POLYLINE POINT 1"',(-37.135,-43.419,0.0));
#709=CARTESIAN_POINT('"POLYLINE POINT 2"',(-35.336999999999996,-44.316,0.0));
#710=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#711,(#712),(#713),$,$);
#711=CIRCLE('"CIRCLE"',#714,5.0);
#712=CARTESIAN_POINT('"TRIM POINT 1"',(-35.336999999999996,-44.316,0.0));
#713=CARTESIAN_POINT('"TRIM POINT 2"',(-34.086999999999996,-44.693,0.0));
#714=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#715,#716,#717);
#715=CARTESIAN_POINT('"CIRCLE CENTER"',(-33.275,-39.760000000000005,0.0));
#716=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#717=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#718=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#719,#720));
#719=CARTESIAN_POINT('"POLYLINE POINT 1"',(-34.086999999999996,-44.693,0.0));
#720=CARTESIAN_POINT('"POLYLINE POINT 2"',(-32.095,-45.016999999999996,0.0));
#721=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#722,(#723),(#724),$,$);
#722=CIRCLE('"CIRCLE"',#725,5.0);
#723=CARTESIAN_POINT('"TRIM POINT 1"',(-32.095,-45.016999999999996,0.0));
#724=CARTESIAN_POINT('"TRIM POINT 2"',(-31.660999999999994,-45.054,0.0));
#725=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#726,#727,#728);
#726=CARTESIAN_POINT('"CIRCLE CENTER"',(-31.455,-40.06,0.0));
#727=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#728=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#729=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#730,#731));
#730=CARTESIAN_POINT('"POLYLINE POINT 1"',(-31.660999999999994,-45.054,0.0));
#731=CARTESIAN_POINT('"POLYLINE POINT 2"',(-24.498999999999995,-45.108000000000004,0.0));
#732=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#733,(#734),(#735),$,$);
#733=CIRCLE('"CIRCLE"',#736,5.0);
#734=CARTESIAN_POINT('"TRIM POINT 1"',(-24.498999999999995,-45.108000000000004,0.0));
#735=CARTESIAN_POINT('"TRIM POINT 2"',(-23.823999999999998,-45.034000000000006,0.0));
#736=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#737,#738,#739);
#737=CARTESIAN_POINT('"CIRCLE CENTER"',(-24.705,-40.11,0.0));
#738=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#739=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#740=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#741,#742));
#741=CARTESIAN_POINT('"POLYLINE POINT 1"',(-23.823999999999998,-45.034000000000006,0.0));
#742=CARTESIAN_POINT('"POLYLINE POINT 2"',(-21.71799999999999,-44.653999999999996,0.0));
#743=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#744,(#745),(#746),$,$);
#744=CIRCLE('"CIRCLE"',#747,5.0);
#745=CARTESIAN_POINT('"TRIM POINT 1"',(-21.71799999999999,-44.653999999999996,0.0));
#746=CARTESIAN_POINT('"TRIM POINT 2"',(-20.491,-44.208,0.0));
#747=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#748,#749,#750);
#748=CARTESIAN_POINT('"CIRCLE CENTER"',(-22.795,-39.769999999999996,0.0));
#749=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#750=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#751=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#752,#753));
#752=CARTESIAN_POINT('"POLYLINE POINT 1"',(-20.491,-44.208,0.0));
#753=CARTESIAN_POINT('"POLYLINE POINT 2"',(-18.86399999999999,-43.358000000000004,0.0));
#754=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#755,(#756),(#757),$,$);
#755=CIRCLE('"CIRCLE"',#758,5.0);
#756=CARTESIAN_POINT('"TRIM POINT 1"',(-18.86399999999999,-43.358000000000004,0.0));
#757=CARTESIAN_POINT('"TRIM POINT 2"',(-17.820999999999998,-42.57299999999999,0.0));
#758=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#759,#760,#761);
#759=CARTESIAN_POINT('"CIRCLE CENTER"',(-21.32499999999999,-39.0,0.0));
#760=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
```

```
#761=DIRECTION('''X DIRECTION''',(1.0,0.0,0.0));
#762=POLYLINE('''POLYLINE FOR CONTOUR: PLANAR_FACE4''',(#763,#764));
#763=CARTESIAN_POINT('''POLYLINE POINT 1''',(-17.820999999999998,-42.57299999999999,0.0));
#764=CARTESIAN_POINT('''POLYLINE POINT 2''',(-16.31899999999999,-40.997,0.0));
#765=TRIMMED_CURVE('''TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4''',#766,(#767),(#768),$,$);
#766=CIRCLE('''CIRCLE''',#769,5.0);
#767=CARTESIAN_POINT('''TRIM POINT 1''',(-16.31899999999999,-40.997,0.0));
#768=CARTESIAN_POINT('''TRIM POINT 2''',(-15.509999999999991,-39.748000000000005,0.0));
#769=AXIS2_PLACEMENT_3D('''CIRCLE PLACEMENT''',#770,#771,#772);
#770=CARTESIAN_POINT('''CIRCLE CENTER''',(-20.064999999999998,-37.68000000000001,0.0));
#771=DIRECTION('''Z DIRECTION''',(0.0,0.0,1.0));
#772=DIRECTION('''X DIRECTION''',(1.0,0.0,0.0));
#773=POLYLINE('''POLYLINE FOR CONTOUR: PLANAR_FACE4''',(#774,#775));
#774=CARTESIAN_POINT('''POLYLINE POINT 1''',(-15.509999999999991,-39.748000000000005,0.0));
#775=CARTESIAN_POINT('''POLYLINE POINT 2''',(-14.748999999999995,-38.06299999999999,0.0));
#776=TRIMMED_CURVE('''TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4''',#777,(#778),(#779),$,$);
#777=CIRCLE('''CIRCLE''',#780,5.0);
#778=CARTESIAN_POINT('''TRIM POINT 1''',(-14.748999999999995,-38.06299999999999,0.0));
#779=CARTESIAN_POINT('''TRIM POINT 2''',(-14.548999999999992,-37.46600000000001,0.0));
#780=AXIS2_PLACEMENT_3D('''CIRCLE PLACEMENT''',#781,#782,#783);
#781=CARTESIAN_POINT('''CIRCLE CENTER''',(-19.375,-36.16999999999999,0.0));
#782=DIRECTION('''Z DIRECTION''',(0.0,0.0,1.0));
#783=DIRECTION('''X DIRECTION''',(1.0,0.0,0.0));
#784=POLYLINE('''POLYLINE FOR CONTOUR: PLANAR_FACE4''',(#785,#786,#787,#788,#789,#790,#791));
#785=CARTESIAN_POINT('''POLYLINE POINT 1''',(-14.548999999999992,-37.46600000000001,0.0));
#786=CARTESIAN_POINT('''POLYLINE POINT 2''',(-5.724999999999994,-4.460000000000008,0.0));
#787=CARTESIAN_POINT('''POLYLINE POINT 2''',(-5.0,-2.8300000000000125,-13.5));
#788=CARTESIAN_POINT('''POLYLINE POINT 3''',(-4.036999999999992,-1.6819999999999888,13.5));
#789=CARTESIAN_POINT('''POLYLINE POINT 4''',(-3.5,-1.2050000000000125,13.5));
#790=CARTESIAN_POINT('''POLYLINE POINT 5''',(-1.9299999999999926,-0.30000000000001137,13.5));
#791=CARTESIAN_POINT('''POLYLINE POINT 6''',(0.0,0.0,-13.5));
#792=AXIS2_PLACEMENT_3D('''''SETUP'' ORIGIN''',#794,#795,#796);
#793=WORKPIECE_SETUP(#3,#797,$,$,());
#794=CARTESIAN_POINT('''''SETUP'': LOCATION''',(0.0,0.0,0.0));
#795=DIRECTION('''AXIS''',(0.0,0.0,1.0));
#796=DIRECTION('''REF_DIRECTION''',(1.0,0.0,0.0));
#797=AXIS2_PLACEMENT_3D('''''WORKPIECE135.0X185.0X40.0'' SETUP''',#798,#799,#800);
#798=CARTESIAN_POINT('''SECURITY PLANE: LOCATION''',(0.0,0.0,0.0));
#799=DIRECTION('''AXIS''',(0.0,0.0,1.0));
#800=DIRECTION('''REF_DIRECTION''',(1.0,0.0,0.0));
ENDSEC;
END-ISO-10303-21
```

9.1.2 Sloped Boundary Test Part

```
ISO-10303-21
HEADER;
FILE_DESCRIPTION(
('Fishhead With 3D Pocket'),
'2;1');
FILE_NAME(
'TEST.STP',
('Wesley Essink','Xianzhi Zhang','Aydin Nassehi'),
('University of Bath'),
' ',
'STEPMAN',
' ');
FILE_SCHEMA(('COMBINED_Schema'));
ENDSEC;
DATA;
#1=PROJECT('''RECOGNISED ISO 14649 PART 21  FILE FROM G&M CODES''',#2,(#3),$,$,$);
#2=WORKPLAN('''MAIN WORKPLAN''',(#4,#5,#6,#7,#8,#9,#10,#11,#12,#13),$,#14,$);
#3=WORKPIECE('''WORKPIECE135.0X185.0X40.0''',$,$,$,$,#24,());
#4=MACHINING_WORKINGSTEP('''WS ''PLANAR_FACE1''''',#15,#16,#17,$);
#5=MACHINING_WORKINGSTEP('''WS ''PLANAR_FACE2''''',#15,#41,#42,$);
#6=MACHINING_WORKINGSTEP('''WS ''PLANAR_FACE3''''',#15,#120,#121,$);
#7=MACHINING_WORKINGSTEP('''WS ''POCKET1''''',#15,#216,#217,$);
#8=MACHINING_WORKINGSTEP('''WS ''POCKET2''''',#15,#289,#290,$);
#9=MACHINING_WORKINGSTEP('''WS ''POCKET3''''',#15,#358,#359,$);
#10=MACHINING_WORKINGSTEP('''WS ''POCKET4''''',#15,#431,#432,$);
#11=MACHINING_WORKINGSTEP('''WS ''POCKET5''''',#15,#500,#501,$);
#12=MACHINING_WORKINGSTEP('''WS ''POCKET6''''',#15,#597,#598,$);
#13=MACHINING_WORKINGSTEP('''WS ''PLANAR_FACE4''''',#15,#618,#619,$);
#14=SETUP('''SETUP''',#792,#15,(#793));
#15=ELEMENTARY_SURFACE('''SECURITY PLANE''',#18);
#16=PLANAR_FACE('''PLANAR_FACE1''',#3,(#17),#22,#23,$,$,$,());
#17=PLANE_MILLING($,$,'''PLANAR_FACE1''',$,$,#29,#30,#31,$,$,$,$,$,$);
#18=AXIS2_PLACEMENT_3D('''SECURITY PLANE PLACEMENT''',#19,#20,#21);
#19=CARTESIAN_POINT('''SECURITY PLANE: LOCATION''',(0.0,0.0,10.0,0.0,0.0));
#20=DIRECTION('''AXIS''',(0.0,0.0,1.0));
#21=DIRECTION('''REF_DIRECTION''',(1.0,0.0,0.0));
#22=AXIS2_PLACEMENT_3D('''PLANAR_FACE1 PLACEMENT''',#34,#35,#36);
#23=ELEMENTARY_SURFACE('''PLANAR_FACE1 DEPTH PLANE''',#37);
#24=BLOCK('''WORKPIECE BLOCK''',#25,135.0,185.0,-40.0);
#25=AXIS2_PLACEMENT_3D('''''WORKPIECE BLOCK'' PLACEMENT''',#26,#27,#28);
#26=CARTESIAN_POINT('''''WORKPIECE BLOCK'': LOCATION''',(0.0,0.0,0.0));
#27=DIRECTION('''AXIS''',(0.0,0.0,1.0));
```

```
#28=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#29=MILLING_CUTTING_TOOL('' T4'',#32,(),$,$,$);
#30=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,33.15,$,$,$,$,$);
#31=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#32=FACEMILL(#33,$,$,$,$);
#33=MILLING_TOOL_DIMENSION(40.0,$,$,$,0.0,$,$);
#34=CARTESIAN_POINT('' PLANAR_FACE1'',(175.0,8.125,0.0));
#35=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#36=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#37=AXIS2_PLACEMENT_3D('' PLANAR_FACE1 DEPTH'',#38,#39,#40);
#38=CARTESIAN_POINT('' PLANAR_FACE1 DEPTH'',(0.0,0.0,-2.0));
#39=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#40=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#41=PLANAR_FACE('' PLANAR_FACE2'',#3,(#42),#43,#44,$,$,$,(#45));
#42=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE2'',$,$,#46,#47,#48,$,$,$,$,$,$,$,$);
#43=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 PLACEMENT'',#51,#52,#53);
#44=ELEMENTARY_SURFACE('' PLANAR_FACE2 DEPTH PLANE'',#54);
#45=BOSS('' PLANAR_FACE2 BOSS'',#3,(),#58,#44,#59,$);
#46=MILLING_CUTTING_TOOL('' T3'',#49,(),$,$,$);
#47=MILLING_TECHNOLOGY(0.013266666666666666,.TCP.,$,132.63333333333333,$,$,$,$,$);
#48=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#49=ENDMILL(#50,$,$,$,$);
#50=MILLING_TOOL_DIMENSION(10.0,$,$,$,0.0,$,$);
#51=CARTESIAN_POINT('' PLANAR_FACE2'',(144.94,112.042,-4.0));
#52=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#53=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#54=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 DEPTH'',#55,#56,#57);
#55=CARTESIAN_POINT('' PLANAR_FACE2 DEPTH'',(-65.0,0.0,-12.0));
#56=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#57=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#58=AXIS2_PLACEMENT_3D('' PLANAR_FACE2 PLACEMENT'',#60,#61,#62);
#59=GENERAL_CLOSED_PROFILE($,#63);
#60=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE2 BOSS"'',(-115.166,20.014000000000001,0.0));
#61=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#62=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#63=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE2'',(#64,#65,#66,#67,#68),.F.);
#64=COMPOSITE_CURVE_SEGMENT($,.T.,#71);
#65=COMPOSITE_CURVE_SEGMENT($,.T.,#75);
#66=COMPOSITE_CURVE_SEGMENT($,.T.,#83);
#67=COMPOSITE_CURVE_SEGMENT($,.T.,#88);
#68=COMPOSITE_CURVE_SEGMENT($,.T.,#96);
#69=DIRECTION('546869734f6e6573466f72596f75', (1.000,0.000,0.000));
#70=DIRECTION('456767416c77617973436f6d6573466973743b29', (1.000,0.000,0.000));
#71=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#72,#73,#74));
#72=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.0,0.0,0.0));
#73=CARTESIAN_POINT('' POLYLINE POINT 2'',(-2.7720000000000056,-25.256000000000014,0.0));
#74=CARTESIAN_POINT('' POLYLINE POINT 2'',(-4.754000000000005,-67.60600000000001,-7.0));
#75=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE2'',#76,(#77),(#78),$,$);
#76=CIRCLE('' CIRCLE'',#79,5.0);
#77=CARTESIAN_POINT('' TRIM POINT 1'',(-4.754000000000005,-67.60600000000001,0.0));
#78=CARTESIAN_POINT('' TRIM POINT 2'',(0.2459999999999951,-72.60600000000001,0.0));
#79=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#80,#81,#82);
#80=CARTESIAN_POINT('' CIRCLE CENTER'',(0.2459999999999951,-67.60600000000001,0.0));
#81=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#82=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#83=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#84,#85,#86,#87));
#84=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.2459999999999951,-72.60600000000001,0.0));
#85=CARTESIAN_POINT('' POLYLINE POINT 2'',(22.676000000000002,-72.60600000000001,0.0));
#86=CARTESIAN_POINT('' POLYLINE POINT 2'',(22.676000000000002,-74.50600000000001,-12.0));
#87=CARTESIAN_POINT('' POLYLINE POINT 3'',(-0.5970000000000084,-74.55200000000002,-7.0));
#88=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE2'',#89,(#90),(#91),$,$);
#89=CIRCLE('' CIRCLE'',#92,5.0);
#90=CARTESIAN_POINT('' TRIM POINT 1'',(-0.5970000000000084,-74.55200000000002,0.0));
#91=CARTESIAN_POINT('' TRIM POINT 2'',(-5.373999999999995,-79.54700000000001,0.0));
#92=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#93,#94,#95);
#93=CARTESIAN_POINT('' CIRCLE CENTER'',(-0.3739999999999952,-79.54600000000002,0.0));
#94=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#95=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#96=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE2'',(#97,#98,#99,#100,#101,#102,#103));
#97=CARTESIAN_POINT('' POLYLINE POINT 1'',(-5.373999999999995,-79.54700000000001,0.0));
#98=CARTESIAN_POINT('' POLYLINE POINT 2'',(-5.373999999999995,-84.608,0.0));
#99=CARTESIAN_POINT('' POLYLINE POINT 2'',(-7.813999999999993,-103.93,-12.0));
#100=CARTESIAN_POINT('' POLYLINE POINT 3'',(-14.774000000000001,-103.93,-7.0));
#101=CARTESIAN_POINT('' POLYLINE POINT 4'',(-14.774000000000001,-49.902000000000015,-12.0));
#102=CARTESIAN_POINT('' POLYLINE POINT 5'',(-6.025999999999996,0.0,-12.0));
#103=CARTESIAN_POINT('' POLYLINE POINT 6'',(0.0,0.0,-7.0));
#120=PLANAR_FACE('' PLANAR_FACE3'',#3,(#121),#122,#123,$,$,$,(#124));
#121=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE3'',$,$,#46,#125,#126,$,$,$,$,$,$,$,$);
#122=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 PLACEMENT'',#127,#128,#129);
#123=ELEMENTARY_SURFACE('' PLANAR_FACE3 DEPTH PLANE'',#130);
#124=BOSS('' PLANAR_FACE3 BOSS'',#3,(),#134,#123,#135,$);
#125=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,127.31666666666666,$,$,$,$,$);
#126=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#127=CARTESIAN_POINT('' PLANAR_FACE3'',(1.694,194.94,-14.0));
#128=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#129=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#130=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 DEPTH'',#131,#132,#133);
#131=CARTESIAN_POINT('' PLANAR_FACE3 DEPTH'',(-11.634,-21.989000000000004,-23.5));
```

```
#132=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#133=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#134=AXIS2_PLACEMENT_3D('' PLANAR_FACE3 PLACEMENT'',#136,#137,#138);
#135=GENERAL_CLOSED_PROFILE($,#139);
#136=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE3 BOSS"'',(99.206,-27.439999999999998,0.0));
#137=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#138=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#139=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE3'',(#140,#141,#142,#143,#144,#145,#146,#147,#148),.F.);
#140=COMPOSITE_CURVE_SEGMENT($,.T.,#149);
#141=COMPOSITE_CURVE_SEGMENT($,.T.,#163);
#142=COMPOSITE_CURVE_SEGMENT($,.T.,#171);
#143=COMPOSITE_CURVE_SEGMENT($,.T.,#174);
#144=COMPOSITE_CURVE_SEGMENT($,.T.,#182);
#145=COMPOSITE_CURVE_SEGMENT($,.T.,#192);
#146=COMPOSITE_CURVE_SEGMENT($,.T.,#200);
#147=COMPOSITE_CURVE_SEGMENT($,.T.,#203);
#148=COMPOSITE_CURVE_SEGMENT($,.T.,#211);
#149=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE3'',(#150,#151,#152,#153,#154,#155,#156,#157,#158,
                                                       #159,#160,#161,#162));
#150=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.0,0.0,0.0));
#151=CARTESIAN_POINT('' POLYLINE POINT 2'',(3.100999999999999,-6.0020000000000095,0.0));
#152=CARTESIAN_POINT('' POLYLINE POINT 2'',(9.549999999999997,-36.75,-15.0));
#153=CARTESIAN_POINT('' POLYLINE POINT 3'',(18.099999999999994,-86.85,-20.0));
#154=CARTESIAN_POINT('' POLYLINE POINT 4'',(18.099999999999994,-152.5,-15.0));
#155=CARTESIAN_POINT('' POLYLINE POINT 5'',(-85.9,-152.5,-15.0));
#156=CARTESIAN_POINT('' POLYLINE POINT 6'',(-85.9,-85.346,-20.0));
#157=CARTESIAN_POINT('' POLYLINE POINT 7'',(-75.9,-28.30000000000001,-15.0));
#158=CARTESIAN_POINT('' POLYLINE POINT 8'',(-69.4,0.0,-20.0));
#159=CARTESIAN_POINT('' POLYLINE POINT 9'',(-66.4,0.0,-15.0));
#160=CARTESIAN_POINT('' POLYLINE POINT 10'',(-71.13300000000001,-35.50200000000001,-15.0));
#161=CARTESIAN_POINT('' POLYLINE POINT 11'',(-73.898,-60.7,-15.0));
#162=CARTESIAN_POINT('' POLYLINE POINT 12'',(-75.88,-103.05000000000001,-15.0));
#163=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3'',#164,(#165),(#166),$,$);
#164=CIRCLE('' CIRCLE'',#167,5.0);
#165=CARTESIAN_POINT('' TRIM POINT 1'',(-75.88,-103.05000000000001,0.0));
#166=CARTESIAN_POINT('' TRIM POINT 2'',(-70.88,-108.05000000000001,0.0));
#167=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#168,#169,#170);
#168=CARTESIAN_POINT('' CIRCLE CENTER'',(-70.88,-103.05000000000001,0.0));
#169=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#170=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#171=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE3'',(#172,#173));
#172=CARTESIAN_POINT('' POLYLINE POINT 1'',(-70.88,-108.05000000000001,0.0));
#173=CARTESIAN_POINT('' POLYLINE POINT 2'',(-41.900000000000006,-108.05000000000001,0.0));
#174=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3'',#175,(#176),(#177),$,$);
#175=CIRCLE('' CIRCLE'',#178,5.0);
#176=CARTESIAN_POINT('' TRIM POINT 1'',(-41.900000000000006,-108.05000000000001,0.0));
#177=CARTESIAN_POINT('' TRIM POINT 2'',(-36.900000000000006,-103.05000000000001,0.0));
#178=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#179,#180,#181);
#179=CARTESIAN_POINT('' CIRCLE CENTER'',(-41.900000000000006,-103.05000000000001,0.0));
#180=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#181=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#182=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE3'',(#183,#184,#185,#186,#187,#188,#189,#190,#191));
#183=CARTESIAN_POINT('' POLYLINE POINT 1'',(-36.900000000000006,-103.05000000000001,0.0));
#184=CARTESIAN_POINT('' POLYLINE POINT 2'',(-36.90200000000001,-65.567,0.0));
#185=CARTESIAN_POINT('' POLYLINE POINT 2'',(-36.91100000000001,-65.435,2.0));
#186=CARTESIAN_POINT('' POLYLINE POINT 3'',(-36.90800000000001,-65.416,-20.0));
#187=CARTESIAN_POINT('' POLYLINE POINT 4'',(-36.932,-65.132,-15.0));
#188=CARTESIAN_POINT('' POLYLINE POINT 5'',(-36.94200000000001,-60.68899999999999,-15.0));
#189=CARTESIAN_POINT('' POLYLINE POINT 6'',(-33.971000000000004,-60.68899999999999,2.0));
#190=CARTESIAN_POINT('' POLYLINE POINT 7'',(-34.94600000000001,-73.05,-15.0));
#191=CARTESIAN_POINT('' POLYLINE POINT 8'',(-34.95,-103.05000000000001,-15.0));
#192=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3'',#193,(#194),(#195),$,$);
#193=CIRCLE('' CIRCLE'',#196,5.0);
#194=CARTESIAN_POINT('' TRIM POINT 1'',(-34.95,-103.05000000000001,0.0));
#195=CARTESIAN_POINT('' TRIM POINT 2'',(-29.950000000000003,-108.05000000000001,0.0));
#196=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#197,#198,#199);
#197=CARTESIAN_POINT('' CIRCLE CENTER'',(-29.950000000000003,-103.05000000000001,0.0));
#198=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#199=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#200=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE3'',(#201,#202));
#201=CARTESIAN_POINT('' POLYLINE POINT 1'',(-29.950000000000003,-108.05000000000001,0.0));
#202=CARTESIAN_POINT('' POLYLINE POINT 2'',(6.941999999999993,-108.04599999999999,0.0));
#203=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE3'',#204,(#205),(#206),$,$);
#204=CIRCLE('' CIRCLE'',#207,5.0);
#205=CARTESIAN_POINT('' TRIM POINT 1'',(6.941999999999993,-108.04599999999999,0.0));
#206=CARTESIAN_POINT('' TRIM POINT 2'',(11.713999999999999,-102.63,0.0));
#207=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#208,#209,#210);
#208=CARTESIAN_POINT('' CIRCLE CENTER'',(6.72999999999999,-103.05000000000001,0.0));
#209=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#210=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#211=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE3'',(#212,#213,#214,#215));
#212=CARTESIAN_POINT('' POLYLINE POINT 1'',(11.713999999999999,-102.63,0.0));
#213=CARTESIAN_POINT('' POLYLINE POINT 2'',(6.547999999999988,-41.982,0.0));
#214=CARTESIAN_POINT('' POLYLINE POINT 2'',(-1.3010000000000002,0.0,-20.0));
#215=CARTESIAN_POINT('' POLYLINE POINT 3'',(0.0,0.0,-15.0));
#216=CLOSED_POCKET('' POCKET1'',#3,(#217),#218,#219,(),$,#220,#221,$,#222);
#217=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET1'',$,$,#46,#223,#224,$,$,$,$,$,$,$,$);
#218=AXIS2_PLACEMENT_3D('' POCKET1 PLACEMENT'',#225,#226,#227);
```

```
#219=ELEMENTARY_SURFACE('"POCKET1 DEPTH PLANE"',#228);
#220=PLANAR_POCKET_BOTTOM_CONDITION();
#221=TOLERANCED_LENGTH_MEASURE(5.0,$);
#222=GENERAL_CLOSED_PROFILE($,#232);
#223=MILLING_TECHNOLOGY(0.008483333333333334,.TCP.,$,106.1,$,$,$,$,$);
#224=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#225=CARTESIAN_POINT('"POCKET1"',(29.924,47.056,-14.0));
#226=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#227=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#228=AXIS2_PLACEMENT_3D('"POCKET1 DEPTH"',#229,#230,#231);
#229=CARTESIAN_POINT('"POCKET1 DEPTH"',(-0.02400000000000091,0.4620000000000033,-17.0));
#230=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#231=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#232=COMPOSITE_CURVE('"BOUNDARY: POCKET1"',(#233,#234,#235,#236,#237,#238,#239,#240,#241),.F.);
#233=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#242);
#234=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#250);
#235=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#253);
#236=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#256);
#237=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#264);
#238=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#267);
#239=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#275);
#240=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#278);
#241=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#286);
#242=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#243,(#244),(#245),.T.,.CARTESIAN.);
#243=CIRCLE('"CIRCLE"',#246,5.0);
#244=CARTESIAN_POINT('"TRIM POINT 1"',(-0.7469999999999999,10.448,0.0));
#245=CARTESIAN_POINT('"TRIM POINT 2"',(-5.524000000000001,5.453000000000003,0.0));
#246=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#247,#248,#249);
#247=CARTESIAN_POINT('"CIRCLE CENTER"',(-0.5240000000000009,5.454000000000001,0.0));
#248=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#249=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#250=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#251,#252));
#251=CARTESIAN_POINT('"POLYLINE POINT 1"',(-5.524000000000001,5.453000000000003,0.0));
#252=CARTESIAN_POINT('"POLYLINE POINT 2"',(-5.524000000000001,0.392000000000003,0.0));
#253=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#254,#255));
#254=CARTESIAN_POINT('"POLYLINE POINT 1"',(-5.524000000000001,0.392000000000003,0.0));
#255=CARTESIAN_POINT('"POLYLINE POINT 2"',(-7.983999999999998,-19.119999999999997,0.0));
#256=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#257,(#258),(#259),.T.,.CARTESIAN.);
#257=CIRCLE('"CIRCLE"',#260,5.0);
#258=CARTESIAN_POINT('"TRIM POINT 1"',(-7.983999999999998,-19.119999999999997,0.0));
#259=CARTESIAN_POINT('"TRIM POINT 2"',(-3.003,-24.555999999999997,0.0));
#260=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#261,#262,#263);
#261=CARTESIAN_POINT('"CIRCLE CENTER"',(-3.003999999999998,-19.555999999999997,0.0));
#262=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#263=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#264=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#265,#266));
#265=CARTESIAN_POINT('"POLYLINE POINT 1"',(-3.003,-24.555999999999997,0.0));
#266=CARTESIAN_POINT('"POLYLINE POINT 2"',(76.876,-24.555999999999997,0.0));
#267=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#268,(#269),(#270),.T.,.CARTESIAN.);
#268=CIRCLE('"CIRCLE"',#271,5.0);
#269=CARTESIAN_POINT('"TRIM POINT 1"',(76.876,-24.555999999999997,0.0));
#270=CARTESIAN_POINT('"TRIM POINT 2"',(81.876,-19.555999999999997,0.0));
#271=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#272,#273,#274);
#272=CARTESIAN_POINT('"CIRCLE CENTER"',(76.876,-19.555999999999997,0.0));
#273=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#274=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#275=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#276,#277));
#276=CARTESIAN_POINT('"POLYLINE POINT 1"',(81.876,-19.555999999999997,0.0));
#277=CARTESIAN_POINT('"POLYLINE POINT 2"',(81.87100000000001,5.631,0.0));
#278=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET1"',#279,(#280),(#281),.T.,.CARTESIAN.);
#279=CIRCLE('"CIRCLE"',#282,5.0);
#280=CARTESIAN_POINT('"TRIM POINT 1"',(81.87100000000001,5.631,0.0));
#281=CARTESIAN_POINT('"TRIM POINT 2"',(77.096,10.410000000000004,0.0));
#282=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#283,#284,#285);
#283=CARTESIAN_POINT('"CIRCLE CENTER"',(76.876,5.4140000000000015,0.0));
#284=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#285=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#286=POLYLINE('"POLYLINE FOR CONTOUR: POCKET1"',(#287,#288));
#287=CARTESIAN_POINT('"POLYLINE POINT 1"',(77.096,10.410000000000004,0.0));
#288=CARTESIAN_POINT('"POLYLINE POINT 2"',(-0.7469999999999999,10.448,0.0));
#289=CLOSED_POCKET('"POCKET2"',#3,(#290),#291,#292,(),$,#293,#294,$,#295);
#290=BOTTOM_AND_SIDE_MILLING($,$,'"POCKET2"',$,$,#46,#296,#297,$,$,$,$,$,$,$,$);
#291=AXIS2_PLACEMENT_3D('"POCKET2 PLACEMENT"',#298,#299,#300);
#292=ELEMENTARY_SURFACE('"POCKET2 DEPTH PLANE"',#301);
#293=PLANAR_POCKET_BOTTOM_CONDITION();
#294=TOLERANCED_LENGTH_MEASURE(5.0,$);
#295=GENERAL_CLOSED_PROFILE($,#305);
#296=MILLING_TECHNOLOGY(0.008483333333333334,.TCP.,$,106.63333333333334,$,$,$,$,$);
#297=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#298=CARTESIAN_POINT('"POCKET2"',(101.797,47.475,-19.0));
#299=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#300=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#301=AXIS2_PLACEMENT_3D('"POCKET2 DEPTH"',#302,#303,#304);
#302=CARTESIAN_POINT('"POCKET2 DEPTH"',(4.503,-14.975000000000001,-24.5));
#303=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#304=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#305=COMPOSITE_CURVE('"BOUNDARY: POCKET2"',(#306,#307,#308,#309,#310,#311,#312,#313),.F.);
#306=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#314);
```

```
#307=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#322);
#308=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#325);
#309=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#333);
#310=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#336);
#311=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#344);
#312=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#347);
#313=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#355);
#314=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET2"',#315,(#316),(#317),.T.,.CARTESIAN.);
#315=CIRCLE('"CIRCLE"',#318,5.0);
#316=CARTESIAN_POINT('"TRIM POINT 1"',(-31.063000000000002,10.015999999999998,0.0));
#317=CARTESIAN_POINT('"TRIM POINT 2"',(-35.842,5.241,0.0));
#318=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#319,#320,#321);
#319=CARTESIAN_POINT('"CIRCLE CENTER"',(-30.846999999999994,5.024999999999999,0.0));
#320=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#321=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#322=POLYLINE('"POLYLINE FOR CONTOUR: POCKET2"',(#323,#324));
#323=CARTESIAN_POINT('"POLYLINE POINT 1"',(-35.842,5.241,0.0));
#324=CARTESIAN_POINT('"POLYLINE POINT 2"',(-35.846999999999994,-19.975,0.0));
#325=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET2"',#326,(#327),(#328),.T.,.CARTESIAN.);
#326=CIRCLE('"CIRCLE"',#329,5.0);
#327=CARTESIAN_POINT('"TRIM POINT 1"',(-35.846999999999994,-19.975,0.0));
#328=CARTESIAN_POINT('"TRIM POINT 2"',(-30.846999999999994,-24.975,0.0));
#329=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#330,#331,#332);
#330=CARTESIAN_POINT('"CIRCLE CENTER"',(-30.846999999999994,-19.975,0.0));
#331=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#332=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#333=POLYLINE('"POLYLINE FOR CONTOUR: POCKET2"',(#334,#335));
#334=CARTESIAN_POINT('"POLYLINE POINT 1"',(-30.846999999999994,-24.975,0.0));
#335=CARTESIAN_POINT('"POLYLINE POINT 2"',(5.003,-24.975,0.0));
#336=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET2"',#337,(#338),(#339),.T.,.CARTESIAN.);
#337=CIRCLE('"CIRCLE"',#340,5.0);
#338=CARTESIAN_POINT('"TRIM POINT 1"',(5.003,-24.975,0.0));
#339=CARTESIAN_POINT('"TRIM POINT 2"',(10.003,-19.975,0.0));
#340=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#341,#342,#343);
#341=CARTESIAN_POINT('"CIRCLE CENTER"',(5.003,-19.975,0.0));
#342=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#343=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#344=POLYLINE('"POLYLINE FOR CONTOUR: POCKET2"',(#345,#346));
#345=CARTESIAN_POINT('"POLYLINE POINT 1"',(10.003,-19.975,0.0));
#346=CARTESIAN_POINT('"POLYLINE POINT 2"',(9.998000000000005,5.211999999999996,0.0));
#347=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET2"',#348,(#349),(#350),.T.,.CARTESIAN.);
#348=CIRCLE('"CIRCLE"',#351,5.0);
#349=CARTESIAN_POINT('"TRIM POINT 1"',(9.998000000000005,5.211999999999996,0.0));
#350=CARTESIAN_POINT('"TRIM POINT 2"',(5.222999999999999,9.991,0.0));
#351=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#352,#353,#354);
#352=CARTESIAN_POINT('"CIRCLE CENTER"',(5.003,4.994999999999997,0.0));
#353=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#354=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#355=POLYLINE('"POLYLINE FOR CONTOUR: POCKET2"',(#356,#357));
#356=CARTESIAN_POINT('"POLYLINE POINT 1"',(5.222999999999999,9.991,0.0));
#357=CARTESIAN_POINT('"POLYLINE POINT 2"',(-31.063000000000002,10.015999999999998,0.0));
#358=CLOSED_POCKET('"POCKET3"',#3,(#359),#360,#361,(),0.785,#362,#363,$,#364);
#359=BOTTOM_AND_SIDE_MILLING($,$,'"POCKET3"',$,$,#46,#365,#366,$,$,$,$,$,0.5,$,$);
#360=AXIS2_PLACEMENT_3D('"POCKET3 PLACEMENT"',#367,#368,#369);
#361=ELEMENTARY_SURFACE('"POCKET3 DEPTH PLANE"',#370);
#362=PLANAR_POCKET_BOTTOM_CONDITION();
#363=TOLERANCED_LENGTH_MEASURE(5.0,$);
#364=GENERAL_CLOSED_PROFILE($,#374);
#365=MILLING_TECHNOLOGY(0.006666666666666667,.TCP.,$,106.1,$,$,$,$,$);
#366=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#367=CARTESIAN_POINT('"POCKET3"',(32.592,32.5,-19.0));
#368=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#369=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#370=AXIS2_PLACEMENT_3D('"POCKET3 DEPTH"',#371,#372,#373);
#371=CARTESIAN_POINT('"POCKET3 DEPTH"',(-2.692,19.509999999999998,-26.5));
#372=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#373=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#374=COMPOSITE_CURVE('"BOUNDARY: POCKET3"',(#375,#376,#377,#378,#379,#380,#381,#382,#383),.F.);
#375=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#384);
#376=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#392);
#377=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#395);
#378=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#398);
#379=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#406);
#380=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#409);
#381=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#417);
#382=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#420);
#383=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#428);
#384=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF POCKET3"',#385,(#386),(#387),.T.,.CARTESIAN.);
#385=CIRCLE('"CIRCLE"',#388,5.0);
#386=CARTESIAN_POINT('"TRIM POINT 1"',(-3.414999999999999,25.003999999999998,0.0));
#387=CARTESIAN_POINT('"TRIM POINT 2"',(-8.192,20.009,0.0));
#388=AXIS2_PLACEMENT_3D('"CIRCLE CENTER"',#389,#390,#391);
#389=CARTESIAN_POINT('"CIRCLE CENTER"',(-3.192,20.009999999999998,0.0));
#390=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#391=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#392=POLYLINE('"POLYLINE FOR CONTOUR: POCKET3"',(#393,#394));
#393=CARTESIAN_POINT('"POLYLINE POINT 1"',(-8.192,20.009,0.0));
#394=CARTESIAN_POINT('"POLYLINE POINT 2"',(-8.192,14.948,0.0));
```

```
#395=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#396,#397));
#396=CARTESIAN_POINT('' POLYLINE POINT 1'',(-8.192,14.948,0.0));
#397=CARTESIAN_POINT('' POLYLINE POINT 2'',(-10.651999999999997,-4.564,0.0));
#398=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#399,(#400),(#401),.T.,.CARTESIAN.);
#399=CIRCLE('' CIRCLE'',#402,5.0);
#400=CARTESIAN_POINT('' TRIM POINT 1'',(-10.651999999999997,-4.564,0.0));
#401=CARTESIAN_POINT('' TRIM POINT 2'',(-5.670999999999999,-10.0,0.0));
#402=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#403,#404,#405);
#403=CARTESIAN_POINT('' CIRCLE CENTER'',(-5.671999999999997,-5.0,0.0));
#404=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#405=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#406=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#407,#408));
#407=CARTESIAN_POINT('' POLYLINE POINT 1'',(-5.670999999999999,-10.0,0.0));
#408=CARTESIAN_POINT('' POLYLINE POINT 2'',(26.408,-10.0,0.0));
#409=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#410,(#411),(#412),.T.,.CARTESIAN.);
#410=CIRCLE('' CIRCLE'',#413,5.0);
#411=CARTESIAN_POINT('' TRIM POINT 1'',(26.408,-10.0,0.0));
#412=CARTESIAN_POINT('' TRIM POINT 2'',(31.408,-5.0,0.0));
#413=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#414,#415,#416);
#414=CARTESIAN_POINT('' CIRCLE CENTER'',(26.408,-5.0,0.0));
#415=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#416=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#417=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#418,#419));
#418=CARTESIAN_POINT('' POLYLINE POINT 1'',(31.408,-5.0,0.0));
#419=CARTESIAN_POINT('' POLYLINE POINT 2'',(31.404000000000003,20.229,0.0));
#420=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET3'',#421,(#422),(#423),.T.,.CARTESIAN.);
#421=CIRCLE('' CIRCLE'',#424,5.0);
#422=CARTESIAN_POINT('' TRIM POINT 1'',(31.404000000000003,20.229,0.0));
#423=CARTESIAN_POINT('' TRIM POINT 2'',(26.429000000000002,25.022,0.0));
#424=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#425,#426,#427);
#425=CARTESIAN_POINT('' CIRCLE CENTER'',(26.408,20.020000000000003,0.0));
#426=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#427=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#428=POLYLINE('' POLYLINE FOR CONTOUR: POCKET3'',(#429,#430));
#429=CARTESIAN_POINT('' POLYLINE POINT 1'',(26.429000000000002,25.022,0.0));
#430=CARTESIAN_POINT('' POLYLINE POINT 2'',(-3.414999999999999,25.003999999999998,0.0));
#431=CLOSED_POCKET('' POCKET4'',#3,(#432),#433,#434,(),$,#435,#436,$,#437);
#432=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET4'',$,$,#46,#438,#439,$,$,$,$,$,$,$,$);
#433=AXIS2_PLACEMENT_3D('' POCKET4 PLACEMENT'',#440,#441,#442);
#434=ELEMENTARY_SURFACE('' POCKET4 DEPTH PLANE'',#443);
#435=PLANAR_POCKET_BOTTOM_CONDITION();
#436=TOLERANCED_LENGTH_MEASURE(5.0,$);
#437=GENERAL_CLOSED_PROFILE($,#447);
#438=MILLING_TECHNOLOGY(0.0138,.TCP.,$,137.93333333333334,$,$,$,$,$);
#439=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#440=CARTESIAN_POINT('' POCKET4'',(55.508,92.56,-25.0));
#441=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#442=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#443=AXIS2_PLACEMENT_3D('' POCKET4 DEPTH'',#444,#445,#446);
#444=CARTESIAN_POINT('' POCKET4 DEPTH'',(-1.5080000000000027,8.739999999999995,-25.0));
#445=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#446=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#447=COMPOSITE_CURVE('' BOUNDARY: POCKET4'',(#448,#449,#450,#451,#452,#453,#454,#455),.F.);
#448=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#456);
#449=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#459);
#450=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#467);
#451=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#470);
#452=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#478);
#453=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#481);
#454=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#489);
#455=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#492);
#456=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#457,#458));
#457=CARTESIAN_POINT('' POLYLINE POINT 1'',(3.4919999999999973,14.239999999999995,0.0));
#458=CARTESIAN_POINT('' POLYLINE POINT 2'',(-23.939000000000004,14.236000000000004,0.0));
#459=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#460,(#461),(#462),.T.,.CARTESIAN.);
#460=CIRCLE('' CIRCLE'',#463,5.0);
#461=CARTESIAN_POINT('' TRIM POINT 1'',(-23.939000000000004,14.236000000000004,0.0));
#462=CARTESIAN_POINT('' TRIM POINT 2'',(-28.732000000000003,9.474000000000004,0.0));
#463=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#464,#465,#466);
#464=CARTESIAN_POINT('' CIRCLE CENTER'',(-23.738000000000003,9.239999999999995,0.0));
#465=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#466=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#467=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#468,#469));
#468=CARTESIAN_POINT('' POLYLINE POINT 1'',(-28.732000000000003,9.474000000000004,0.0));
#469=CARTESIAN_POINT('' POLYLINE POINT 2'',(-30.488000000000003,-28.11,0.0));
#470=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#471,(#472),(#473),.T.,.CARTESIAN.);
#471=CIRCLE('' CIRCLE'',#474,5.0);
#472=CARTESIAN_POINT('' TRIM POINT 1'',(-30.488000000000003,-28.11,0.0));
#473=CARTESIAN_POINT('' TRIM POINT 2'',(-25.488000000000003,-33.11,0.0));
#474=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#475,#476,#477);
#475=CARTESIAN_POINT('' CIRCLE CENTER'',(-25.488000000000003,-28.11,0.0));
#476=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#477=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#478=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#479,#480));
#479=CARTESIAN_POINT('' POLYLINE POINT 1'',(-25.488000000000003,-33.11,0.0));
#480=CARTESIAN_POINT('' POLYLINE POINT 2'',(3.4919999999999973,-33.11,0.0));
#481=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#482,(#483),(#484),.T.,.CARTESIAN.);
#482=CIRCLE('' CIRCLE'',#485,5.0);
```

```
#483=CARTESIAN_POINT('' TRIM POINT 1'',(3.4919999999999973,-33.11,0.0));
#484=CARTESIAN_POINT('' TRIM POINT 2'',(8.491999999999997,-28.11,0.0));
#485=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#486,#487,#488);
#486=CARTESIAN_POINT('' CIRCLE CENTER'',(3.4919999999999973,-28.11,0.0));
#487=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#488=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#489=POLYLINE('' POLYLINE FOR CONTOUR: POCKET4'',(#490,#491));
#490=CARTESIAN_POINT('' POLYLINE POINT 1'',(8.491999999999997,-28.11,0.0));
#491=CARTESIAN_POINT('' POLYLINE POINT 2'',(8.491999999999997,9.239999999999995,0.0));
#492=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET4'',#493,(#494),(#495),.T.,.CARTESIAN.);
#493=CIRCLE('' CIRCLE'',#496,5.0);
#494=CARTESIAN_POINT('' TRIM POINT 1'',(8.491999999999997,9.239999999999995,0.0));
#495=CARTESIAN_POINT('' TRIM POINT 2'',(3.4919999999999973,14.239999999999995,0.0));
#496=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#497,#498,#499);
#497=CARTESIAN_POINT('' CIRCLE CENTER'',(3.4919999999999973,9.239999999999995,0.0));
#498=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#499=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#500=CLOSED_POCKET('' POCKET5'',#3,(#501),#502,#503,(#504),$,#505,#506,$,#507);
#501=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET5'',$,$,#46,#508,#509,$,$,$,$,$,$,$);
#502=AXIS2_PLACEMENT_3D('' POCKET5 PLACEMENT'',#510,#511,#512);
#503=ELEMENTARY_SURFACE('' POCKET5 DEPTH PLANE'',#513);
#504=BOSS('' POCKET5_BOSS'',#3,(),#517,#503,#518,$);
#505=PLANAR_POCKET_BOTTOM_CONDITION();
#506=TOLERANCED_LENGTH_MEASURE(5.0,$);
#507=GENERAL_CLOSED_PROFILE($,#527);
#508=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,107.16666666666667,$,$,$,$,$);
#509=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#510=CARTESIAN_POINT('' POCKET5'',(104.64,87.373,-25.5));
#511=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#512=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#513=AXIS2_PLACEMENT_3D('' POCKET5 DEPTH'',#514,#515,#516);
#514=CARTESIAN_POINT('' POCKET5 DEPTH'',(0.8529999999999944,-4.927000000000007,-25.5));
#515=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#516=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#517=AXIS2_PLACEMENT_3D('' POCKET5 PLACEMENT'',#519,#520,#521);
#518=CIRCULAR_CLOSED_PROFILE(#522,#523);
#519=CARTESIAN_POINT('' LOCATIONPOCKET5'',(-18.040000000000006,-3.3230000000000075,-12.0));
#520=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#521=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#522=AXIS2_PLACEMENT_3D('' CIRCULAR PROFILE LOCATION'',#524,#525,#526);
#523=TOLERANCED_LENGTH_MEASURE(8.5,$);
#524=CARTESIAN_POINT('' LOCATION POINT'',(0.0,0.0,0.0));
#525=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#526=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#527=COMPOSITE_CURVE('' BOUNDARY: POCKET5'',(#528,#529,#530,#531,#532,#533,#534,#535,#536,#537,#538),.F.);
#528=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#539);
#529=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#547);
#530=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#550);
#531=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#558);
#532=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#561);
#533=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#564);
#534=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#572);
#535=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#575);
#536=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#583);
#537=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#586);
#538=COMPOSITE_CURVE_SEGMENT(.CONTINUOUS.,.T.,#594);
#539=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#540,(#541),(#542),.T.,.CARTESIAN.);
#540=CIRCLE('' CIRCLE'',#543,5.0);
#541=CARTESIAN_POINT('' TRIM POINT 1'',(-32.947,18.429000000000002,0.0));
#542=CARTESIAN_POINT('' TRIM POINT 2'',(-37.995000000000005,13.939999999999998,0.0));
#543=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#544,#545,#546);
#544=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.019999999999996,13.426999999999992,0.0));
#545=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#546=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#547=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#548,#549));
#548=CARTESIAN_POINT('' POLYLINE POINT 1'',(-37.995000000000005,13.939999999999998,0.0));
#549=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.022000000000006,13.527000000000001,0.0));
#550=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#551,(#552),(#553),.T.,.CARTESIAN.);
#551=CIRCLE('' CIRCLE'',#554,5.0);
#552=CARTESIAN_POINT('' TRIM POINT 1'',(-38.022000000000006,13.527000000000001,0.0));
#553=CARTESIAN_POINT('' TRIM POINT 2'',(-38.092,13.096999999999994,0.0));
#554=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#555,#556,#557);
#555=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.129999999999995,12.506999999999999,0.0));
#556=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#557=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#558=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#559,#560));
#559=CARTESIAN_POINT('' POLYLINE POINT 1'',(-38.092,13.096999999999994,0.0));
#560=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.566,7.0719999999999885,0.0));
#561=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#562,#563));
#562=CARTESIAN_POINT('' POLYLINE POINT 1'',(-38.566,7.0719999999999885,0.0));
#563=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.57000000000001,-22.803000000000001,0.0));
#564=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#565,(#566),(#567),.T.,.CARTESIAN.);
#565=CIRCLE('' CIRCLE'',#568,5.0);
#566=CARTESIAN_POINT('' TRIM POINT 1'',(-38.57000000000001,-22.803000000000001,0.0));
#567=CARTESIAN_POINT('' TRIM POINT 2'',(-33.57000000000001,-27.803000000000004,0.0));
#568=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#569,#570,#571);
#569=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.57000000000001,-22.803000000000001,0.0));
#570=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
```

```
#571=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#572=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#573,#574));
#573=CARTESIAN_POINT('' POLYLINE POINT 1'',(-33.57000000000001,-27.803000000000004,0.0));
#574=CARTESIAN_POINT('' POLYLINE POINT 2'',(3.0720000000000027,-27.799000000000007,0.0));
#575=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#576,(#577),(#578),.T.,.CARTESIAN.);
#576=CIRCLE('' CIRCLE'',#579,5.0);
#577=CARTESIAN_POINT('' TRIM POINT 1'',(3.0720000000000027,-27.799000000000007,0.0));
#578=CARTESIAN_POINT('' TRIM POINT 2'',(7.843999999999994,-22.38300000000001,0.0));
#579=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#580,#581,#582);
#580=CARTESIAN_POINT('' CIRCLE CENTER'',(2.8599999999999994,-22.80300000000001,0.0));
#581=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#582=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#583=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#584,#585));
#584=CARTESIAN_POINT('' POLYLINE POINT 1'',(7.843999999999994,-22.38300000000001,0.0));
#585=CARTESIAN_POINT('' POLYLINE POINT 2'',(4.810999999999993,13.549999999999997,0.0));
#586=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF POCKET5'',#587,(#588),(#589),.T.,.CARTESIAN.);
#587=CIRCLE('' CIRCLE'',#590,5.0);
#588=CARTESIAN_POINT('' TRIM POINT 1'',(4.810999999999993,13.549999999999997,0.0));
#589=CARTESIAN_POINT('' TRIM POINT 2'',(-0.07800000000000296,17.94099999999999,0.0));
#590=AXIS2_PLACEMENT_3D('' CIRCLE CENTER'',#591,#592,#593);
#591=CARTESIAN_POINT('' CIRCLE CENTER'',(-0.15000000000000568,12.946999999999989,0.0));
#592=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#593=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#594=POLYLINE('' POLYLINE FOR CONTOUR: POCKET5'',(#595,#596));
#595=CARTESIAN_POINT('' POLYLINE POINT 1'',(-0.07800000000000296,17.94099999999999,0.0));
#596=CARTESIAN_POINT('' POLYLINE POINT 2'',(-32.947,18.429000000000002,0.0));
#597=CLOSED_POCKET('' POCKET6'',#3,(#598),#599,#600,(),$,#601,#602,$,#603);
#598=BOTTOM_AND_SIDE_MILLING($,$,'' POCKET6'',$,$,#46,#604,#605,$,$,$,$,$,$,$);
#599=AXIS2_PLACEMENT_3D('' POCKET6 PLACEMENT'',#606,#607,#608);
#600=ELEMENTARY_SURFACE('' POCKET6 DEPTH PLANE'',#609);
#601=PLANAR_POCKET_BOTTOM_CONDITION();
#602=TOLERANCED_LENGTH_MEASURE(5.0,$);
#603=CIRCULAR_CLOSED_PROFILE(#613,#614);
#604=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,95.5,$,$,$,$,$);
#605=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#606=CARTESIAN_POINT('' POCKET6'',(86.6,84.055,-25.5));
#607=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#608=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#609=AXIS2_PLACEMENT_3D('' POCKET6 DEPTH'',#610,#611,#612);
#610=CARTESIAN_POINT('' POCKET6 DEPTH'',(0.0,0.0,-27.0));
#611=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#612=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#613=AXIS2_PLACEMENT_3D('' CIRCULAR PROFILE LOCATION'',#615,#616,#617);
#614=TOLERANCED_LENGTH_MEASURE(14.0,$);
#615=CARTESIAN_POINT('' LOCATION POINT'',(0.0,0.0,0.0));
#616=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#617=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#618=PLANAR_FACE('' PLANAR_FACE4'',#3,(#619),#620,#621,$,$,$,(#622));
#619=BOTTOM_AND_SIDE_MILLING($,$,'' PLANAR_FACE4'',$,$,#46,#623,#624,$,$,$,$,$,$,$);
#620=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 PLACEMENT'',#625,#626,#627);
#621=ELEMENTARY_SURFACE('' PLANAR_FACE4 DEPTH PLANE'',#628);
#622=BOSS('' PLANAR_FACE4 BOSS'',#3,(),#632,#621,#633,$);
#623=MILLING_TECHNOLOGY(0.004166666666666667,.TCP.,$,108.75,$,$,$,$,$);
#624=MILLING_MACHINE_FUNCTIONS(.F.,$,$,$,$,(),$,$,$,());
#625=CARTESIAN_POINT('' PLANAR_FACE4'',(1.694,194.94,-25.5));
#626=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#627=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#628=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 DEPTH'',#629,#630,#631);
#629=CARTESIAN_POINT('' PLANAR_FACE4 DEPTH'',(-11.634,-21.989000000000004,-27.0));
#630=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#631=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#632=AXIS2_PLACEMENT_3D('' PLANAR_FACE4 PLACEMENT'',#634,#635,#636);
#633=GENERAL_CLOSED_PROFILE($,#637);
#634=CARTESIAN_POINT('' LOCATION: "PLANAR_FACE4 BOSS"'',(91.31099999999999,-27.439999999999998,0.0));
#635=DIRECTION('' AXIS'',(0.0,0.0,1.0));
#636=DIRECTION('' REF_DIRECTION'',(1.0,0.0,0.0));
#637=COMPOSITE_CURVE('' BOUNDARY: PLANAR_FACE4'',(#638,#639,#640,#641,#642,#643,#644,#645,#646,#647,
                                                  #648,#649,#650,#651,#652,#653,#654,#655,#656,#657,#658),.F.);
#638=COMPOSITE_CURVE_SEGMENT($,.T.,#659);
#639=COMPOSITE_CURVE_SEGMENT($,.T.,#676);
#640=COMPOSITE_CURVE_SEGMENT($,.T.,#684);
#641=COMPOSITE_CURVE_SEGMENT($,.T.,#687);
#642=COMPOSITE_CURVE_SEGMENT($,.T.,#695);
#643=COMPOSITE_CURVE_SEGMENT($,.T.,#699);
#644=COMPOSITE_CURVE_SEGMENT($,.T.,#707);
#645=COMPOSITE_CURVE_SEGMENT($,.T.,#710);
#646=COMPOSITE_CURVE_SEGMENT($,.T.,#718);
#647=COMPOSITE_CURVE_SEGMENT($,.T.,#721);
#648=COMPOSITE_CURVE_SEGMENT($,.T.,#729);
#649=COMPOSITE_CURVE_SEGMENT($,.T.,#732);
#650=COMPOSITE_CURVE_SEGMENT($,.T.,#740);
#651=COMPOSITE_CURVE_SEGMENT($,.T.,#743);
#652=COMPOSITE_CURVE_SEGMENT($,.T.,#751);
#653=COMPOSITE_CURVE_SEGMENT($,.T.,#754);
#654=COMPOSITE_CURVE_SEGMENT($,.T.,#762);
#655=COMPOSITE_CURVE_SEGMENT($,.T.,#765);
#656=COMPOSITE_CURVE_SEGMENT($,.T.,#773);
#657=COMPOSITE_CURVE_SEGMENT($,.T.,#776);
```

```
#658=COMPOSITE_CURVE_SEGMENT($,.T.,#784);
#659=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#660,#661,#662,#663,#664,#665,#666,#667,#668,
                                                        #669,#670,#671,#672,#673,#674,#675));
#660=CARTESIAN_POINT('' POLYLINE POINT 1'',(0.0,0.0,0.0));
#661=CARTESIAN_POINT('' POLYLINE POINT 2'',(7.895000000000001,0.0,0.0));
#662=CARTESIAN_POINT('' POLYLINE POINT 2'',(10.996000000000001,-6.0020000000000095,13.5));
#663=CARTESIAN_POINT('' POLYLINE POINT 3'',(17.445000000000007,-36.75,-13.5));
#664=CARTESIAN_POINT('' POLYLINE POINT 4'',(25.995000000000005,-86.85,13.5));
#665=CARTESIAN_POINT('' POLYLINE POINT 5'',(25.995000000000005,-152.5,-13.5));
#666=CARTESIAN_POINT('' POLYLINE POINT 6'',(-78.005,-152.5,-13.5));
#667=CARTESIAN_POINT('' POLYLINE POINT 7'',(-78.005,-85.346,13.5));
#668=CARTESIAN_POINT('' POLYLINE POINT 8'',(-68.005,-28.30000000000001,-13.5));
#669=CARTESIAN_POINT('' POLYLINE POINT 9'',(-61.504999999999995,0.0,13.5));
#670=CARTESIAN_POINT('' POLYLINE POINT 10'',(-58.504999999999995,0.0,-13.5));
#671=CARTESIAN_POINT('' POLYLINE POINT 11'',(-54.504999999999995,-0.25,13.5));
#672=CARTESIAN_POINT('' POLYLINE POINT 12'',(-52.629999999999995,-1.0999999999999943,13.5));
#673=CARTESIAN_POINT('' POLYLINE POINT 13'',(-51.254999999999995,-2.4989999999999952,-13.5));
#674=CARTESIAN_POINT('' POLYLINE POINT 14'',(-50.404999999999994,-4.349999999999994,-13.5));
#675=CARTESIAN_POINT('' POLYLINE POINT 15'',(-41.55799999999999,-37.43700000000001,-13.5));
#676=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#677,(#678),(#679),$,$);
#677=CIRCLE('' CIRCLE'',#680,5.0);
#678=CARTESIAN_POINT('' TRIM POINT 1'',(-41.55799999999999,-37.43700000000001,0.0));
#679=CARTESIAN_POINT('' TRIM POINT 2'',(-41.340999999999994,-38.01599999999999,0.0));
#680=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#681,#682,#683);
#681=CARTESIAN_POINT('' CIRCLE CENTER'',(-36.775,-35.97,0.0));
#682=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#683=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#684=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#685,#686));
#685=CARTESIAN_POINT('' POLYLINE POINT 1'',(-41.340999999999994,-38.01599999999999,0.0));
#686=CARTESIAN_POINT('' POLYLINE POINT 2'',(-40.474,-39.943,0.0));
#687=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#688,(#689),(#690),$,$);
#688=CIRCLE('' CIRCLE'',#691,5.0);
#689=CARTESIAN_POINT('' TRIM POINT 1'',(-40.474,-39.943,0.0));
#690=CARTESIAN_POINT('' TRIM POINT 2'',(-39.751,-41.031000000000006,0.0));
#691=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#692,#693,#694);
#692=CARTESIAN_POINT('' CIRCLE CENTER'',(-35.98499999999999,-37.74000000000001,0.0));
#693=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#694=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#695=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#696,#697,#698));
#696=CARTESIAN_POINT('' POLYLINE POINT 1'',(-39.751,-41.031000000000006,0.0));
#697=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.69199999999999,-42.163,0.0));
#698=CARTESIAN_POINT('' POLYLINE POINT 2'',(-38.19499999999999,-42.656000000000006,-13.5));
#699=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#700,(#701),(#702),$,$);
#700=CIRCLE('' CIRCLE'',#703,5.0);
#701=CARTESIAN_POINT('' TRIM POINT 1'',(-38.19499999999999,-42.656000000000006,0.0));
#702=CARTESIAN_POINT('' TRIM POINT 2'',(-37.135,-43.419,0.0));
#703=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#704,#705,#706);
#704=CARTESIAN_POINT('' CIRCLE CENTER'',(-34.76499999999999,-39.00999999999999,0.0));
#705=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#706=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#707=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#708,#709));
#708=CARTESIAN_POINT('' POLYLINE POINT 1'',(-37.135,-43.419,0.0));
#709=CARTESIAN_POINT('' POLYLINE POINT 2'',(-35.336999999999996,-44.316,0.0));
#710=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#711,(#712),(#713),$,$);
#711=CIRCLE('' CIRCLE'',#714,5.0);
#712=CARTESIAN_POINT('' TRIM POINT 1'',(-35.336999999999996,-44.316,0.0));
#713=CARTESIAN_POINT('' TRIM POINT 2'',(-34.086999999999996,-44.693,0.0));
#714=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#715,#716,#717);
#715=CARTESIAN_POINT('' CIRCLE CENTER'',(-33.275,-39.760000000000005,0.0));
#716=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#717=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#718=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#719,#720));
#719=CARTESIAN_POINT('' POLYLINE POINT 1'',(-34.086999999999996,-44.693,0.0));
#720=CARTESIAN_POINT('' POLYLINE POINT 2'',(-32.095,-45.016999999999996,0.0));
#721=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#722,(#723),(#724),$,$);
#722=CIRCLE('' CIRCLE'',#725,5.0);
#723=CARTESIAN_POINT('' TRIM POINT 1'',(-32.095,-45.016999999999996,0.0));
#724=CARTESIAN_POINT('' TRIM POINT 2'',(-31.660999999999994,-45.054,0.0));
#725=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#726,#727,#728);
#726=CARTESIAN_POINT('' CIRCLE CENTER'',(-31.455,-40.06,0.0));
#727=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#728=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#729=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#730,#731));
#730=CARTESIAN_POINT('' POLYLINE POINT 1'',(-31.660999999999994,-45.054,0.0));
#731=CARTESIAN_POINT('' POLYLINE POINT 2'',(-24.498999999999995,-45.108000000000004,0.0));
#732=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#733,(#734),(#735),$,$);
#733=CIRCLE('' CIRCLE'',#736,5.0);
#734=CARTESIAN_POINT('' TRIM POINT 1'',(-24.498999999999995,-45.108000000000004,0.0));
#735=CARTESIAN_POINT('' TRIM POINT 2'',(-23.823999999999998,-45.034000000000006,0.0));
#736=AXIS2_PLACEMENT_3D('' CIRCLE PLACEMENT'',#737,#738,#739);
#737=CARTESIAN_POINT('' CIRCLE CENTER'',(-24.705,-40.11,0.0));
#738=DIRECTION('' Z DIRECTION'',(0.0,0.0,1.0));
#739=DIRECTION('' X DIRECTION'',(1.0,0.0,0.0));
#740=POLYLINE('' POLYLINE FOR CONTOUR: PLANAR_FACE4'',(#741,#742));
#741=CARTESIAN_POINT('' POLYLINE POINT 1'',(-23.823999999999998,-45.034000000000006,0.0));
#742=CARTESIAN_POINT('' POLYLINE POINT 2'',(-21.71799999999999,-44.653999999999996,0.0));
#743=TRIMMED_CURVE('' TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4'',#744,(#745),(#746),$,$);
#744=CIRCLE('' CIRCLE'',#747,5.0);
```

```
#745=CARTESIAN_POINT('"TRIM POINT 1"',(-21.71799999999999,-44.653999999999996,0.0));
#746=CARTESIAN_POINT('"TRIM POINT 2"',(-20.491,-44.208,0.0));
#747=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#748,#749,#750);
#748=CARTESIAN_POINT('"CIRCLE CENTER"',(-22.795,-39.769999999999996,0.0));
#749=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#750=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#751=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#752,#753));
#752=CARTESIAN_POINT('"POLYLINE POINT 1"',(-20.491,-44.208,0.0));
#753=CARTESIAN_POINT('"POLYLINE POINT 2"',(-18.86399999999999,-43.358000000000004,0.0));
#754=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#755,(#756),(#757),$,$);
#755=CIRCLE('"CIRCLE"',#758,5.0);
#756=CARTESIAN_POINT('"TRIM POINT 1"',(-18.86399999999999,-43.358000000000004,0.0));
#757=CARTESIAN_POINT('"TRIM POINT 2"',(-17.820999999999998,-42.57299999999999,0.0));
#758=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#759,#760,#761);
#759=CARTESIAN_POINT('"CIRCLE CENTER"',(-21.32499999999999,-39.0,0.0));
#760=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#761=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#762=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#763,#764));
#763=CARTESIAN_POINT('"POLYLINE POINT 1"',(-17.820999999999998,-42.57299999999999,0.0));
#764=CARTESIAN_POINT('"POLYLINE POINT 2"',(-16.31899999999999,-40.997,0.0));
#765=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#766,(#767),(#768),$,$);
#766=CIRCLE('"CIRCLE"',#769,5.0);
#767=CARTESIAN_POINT('"TRIM POINT 1"',(-16.31899999999999,-40.997,0.0));
#768=CARTESIAN_POINT('"TRIM POINT 2"',(-15.509999999999991,-39.748000000000005,0.0));
#769=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#770,#771,#772);
#770=CARTESIAN_POINT('"CIRCLE CENTER"',(-20.064999999999998,-37.68000000000001,0.0));
#771=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#772=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#773=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#774,#775));
#774=CARTESIAN_POINT('"POLYLINE POINT 1"',(-15.509999999999991,-39.748000000000005,0.0));
#775=CARTESIAN_POINT('"POLYLINE POINT 2"',(-14.748999999999995,-38.06299999999999,0.0));
#776=TRIMMED_CURVE('"TRIMMED CURVE FOR CONTOUR OF PLANAR_FACE4"',#777,(#778),(#779),$,$);
#777=CIRCLE('"CIRCLE"',#780,5.0);
#778=CARTESIAN_POINT('"TRIM POINT 1"',(-14.748999999999995,-38.06299999999999,0.0));
#779=CARTESIAN_POINT('"TRIM POINT 2"',(-14.548999999999992,-37.46600000000001,0.0));
#780=AXIS2_PLACEMENT_3D('"CIRCLE PLACEMENT"',#781,#782,#783);
#781=CARTESIAN_POINT('"CIRCLE CENTER"',(-19.375,-36.16999999999999,0.0));
#782=DIRECTION('"Z DIRECTION"',(0.0,0.0,1.0));
#783=DIRECTION('"X DIRECTION"',(1.0,0.0,0.0));
#784=POLYLINE('"POLYLINE FOR CONTOUR: PLANAR_FACE4"',(#785,#786,#787,#788,#789,#790,#791));
#785=CARTESIAN_POINT('"POLYLINE POINT 1"',(-14.548999999999992,-37.46600000000001,0.0));
#786=CARTESIAN_POINT('"POLYLINE POINT 2"',(-5.724999999999994,-4.460000000000008,0.0));
#787=CARTESIAN_POINT('"POLYLINE POINT 2"',(-5.0,-2.8300000000000125,-13.5));
#788=CARTESIAN_POINT('"POLYLINE POINT 3"',(-4.036999999999992,-1.6819999999999988,13.5));
#789=CARTESIAN_POINT('"POLYLINE POINT 4"',(-3.5,-1.2050000000000125,13.5));
#790=CARTESIAN_POINT('"POLYLINE POINT 5"',(-1.9299999999999926,-0.30000000000001137,13.5));
#791=CARTESIAN_POINT('"POLYLINE POINT 6"',(0.0,0.0,-13.5));
#792=AXIS2_PLACEMENT_3D('""SETUP" ORIGIN"',#794,#795,#796);
#793=WORKPIECE_SETUP(#3,#797,$,$,());
#794=CARTESIAN_POINT('""SETUP": LOCATION"',(0.0,0.0,0.0));
#795=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#796=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#797=AXIS2_PLACEMENT_3D('""WORKPIECE135.0X185.0X40.0" SETUP"',#798,#799,#800);
#798=CARTESIAN_POINT('"SECURITY PLANE: LOCATION"',(0.0,0.0,0.0));
#799=DIRECTION('"AXIS"',(0.0,0.0,1.0));
#800=DIRECTION('"REF_DIRECTION"',(1.0,0.0,0.0));
#801=DIRECTION('494c4f5645594f5552414348454c', (1.000,0.000,0.000));
ENDSEC;
END-ISO-10303-21
```