Evolutionary Approaches to Robot Path Planning

A thesis submitted for the degree of

Doctor of Philosophy

by

Simon Kent

Department of Information Systems and Computing,
Brunel University,
Uxbridge, Middlesex, UB8 3PH, United Kingdom.

March 1999

Abstract

The ultimate goal in robotics is to create machines which are more independent and rely less on humans to guide them in their operation. There are many sub-systems which may be present in such a robot, one of which is path planning — the ability to determine a sequence of positions or configurations between an initial and goal position within a particular obstacle cluttered workspace.

Many classical path planning techniques have been developed, but these tend to have drawbacks such as their computational requirements; the suitability of the plans they produce for a particular application; or how well they are able to generalise to unseen problems.

In recent years, evolutionary based problem solving techniques have seen a rise in popularity, possibly coinciding with the improvement in the computational power afforded researches by successful developments in hardware. These techniques adopt some of the features of natural evolution and mimic them in a computer. The increase in the number of publications in the areas of Genetic Algorithms (GA) and Genetic Programming (GP) demonstrate the success achieved when applying these techniques to ever more problem areas.

This dissertation presents research conducted to determine whether there is a place for Evolutionary Approaches, and specifically GA and GP, in the development of future path planning techniques.

Contents

List of Tables

List of Figures

Acknowledgements

Thank you to Dimitris, my supervisor, for all his contributions, and to Professor Ray Paul for his invaluable advice. I would also like to acknowledge the support of the EPSRC who provided the funding for my research.

I have appreciated the support and advice of all my friends and colleagues at Brunel, and further afield, and for the opportunity to draw upon the variety of knowledge they have had to offer.

Thanks to Alison for all she has done for me, and especially for showing me how, eventually, to complete a dissertation.

Many thanks to my family, and especially my parents, for their continued support throughout the duration of the research carried out for this dissertation.

Diolch i chi Gymru am gynug y sefyllfa berfaith i fi orffen fy narlith.

Publications arising from this dissertation

ELLIOT, C., KENT, S., HAMMOND, P., DRACOPOULOS, D., DOWNER, M. AND SPEIGHT, P., [1997]. A comparison of artificial intelligence techniques for the identification of people at high risk of oral cancer. In *Proceedings of the 44th Annual General Meeting of the British Society for Dental Research*, p. 275, Brighton, UK.

ELLIOTT, C., KENT, S., HAMMOND, P., DRACOPOULOS, D., DOWNER, M. AND SPEIGHT, P., [1997]. "A comparison of artificial intelligence techniques for the identification of people at high risk of oral cancer." *Journal of Dental Research*, **76**(5), 1053.

KENT, S., [1996]. *Diagnosis of Oral Cancer using Genetic Programming*. Tech. Rep. CSTR-96-14 ; CNES-96-04, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK.

KENT, S. AND DRACOPOULOS, D., [1997]. "Genetic programming for prediction and control." *Neural Computing and Applications*, **6**(4), 214–228.

KENT, S. AND DRACOPOULOS, D. C., [1996]. "Bulk synchronous parallelisation of genetic programming." In Waśniewski, J. (ed.), *Applied parallel computing : industrial strength computation and optimization ; Proceedings of the third International Workshop, PARA '96*, pp. 216–226, Berlin, Germany. Springer Verlag.

KENT, S. AND DRACOPOULOS, D. C., [1996]. "Speeding up genetic programming: A parallel BSP implementation." In Koza, J. R., Goldberg, D. E., Fogel, D. B. and Riolo, R. L. (eds.), *Genetic Programming 1996: Proceedings of the First Conference*, p. 421, Cambridge, MA, USA. MIT Press.

The Three Laws of Robotics

1. A robot may not injure a human being, or through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Handbook of Robotics
56th Edition, 2058 AD

Isaac Asimov (I, Robot)

1

Thesis Introduction

1.1 Introduction

Computers have shown themselves to be extremely capable in many application areas, and have transformed the world in which we live. Their application has helped to enhance the quality of traditionally human tasks, and has completely automated other tasks, replacing the need for humans to carry them out. However, automated systems controlled by computers cannot beat man in all areas.

One such area in which computers fall short of humans is in their ability to plan accurate paths for the movements which a robot should perform to complete a task. The tasks can be very varied, for example transporting a package around a warehouse, placing a bolt in a hole on a production line, or pouring water from a kettle into a cup to make a cup of tea. In all of these cases, a common problem is present which involves generating a sequence of positions in the workspace allowing a robot to traverse, in as few moves as possible, the route from an initial to a goal position whilst avoiding any obstacles present.

It is not being suggested categorically that computers cannot plan paths, but that the techniques currently available do not enable them to exhibit the combined speed of planning, accuracy, and ability to generalise which is present in

humans. Much research has been carried out on the problems of motion planning, particularly in the 1980's, but few of these methods are simple enough and powerful enough to be used for practical robotics [Lozano-Pérez, 1987]

This is borne out by the lack of robots which can match humans in their ability to ski, play squash or drive, let alone perform all of these. Admittedly a robot which could perform even moderately at any of these abilities would require more than path planning alone, nevertheless competent path planning is an essential component. This dissertation examines the existing methods used to perform path planning, and seeks ways by which they might be improved in the future.

This introductory chapter describes the need for motion planning. It considers the conventional way which computers are used to solve problems, such as path planning, and how alternative problem solving techniques, using evolutionary computation can overcome some of the drawbacks associated with a more conventional approach. The chapter concludes with a summary outlining the way in which existing path planning techniques will be investigated during the remainder of this thesis.

1.2 The need for Path Planning

There is an increasing need for proficient path planning systems. Use is made of mechanical devices which improve the efficiency and safety of the manual work traditionally carried out by humans or animals. Robots have been used for several years in industrial assembly plants. These robots move components into place, weld and bolt them together, and perform many of the functions which would previously have required large manual work forces. These robots are controlled by programs defining the specific movements which must be performed in order to achieve a goal. A change in the goal, for instance the introduction of a design change for a new model of car, would require expensive reprogramming using a robot control language [Sheu and Xue, 1993,

Chapter 3]. It is suggested that a better approach is to build intelligent robot systems which when provided with a goal, can establish for themselves the set of positions which must be followed to achieve the goal [Sheu and Xue, 1993, Chapter 1].

Historically, machines which have been used by humans to perform generalised, rather than specific, repetitive tasks have not performed their own path planning. Examples exist of agricultural machinery, which can be applied, by humans, to a miscellany of tasks. These have traditionally required their human operators to perform the complex path planning necessary for each task rather than operators relying on the equipment to plan for itself. Auxiliary computer control systems have become commonplace in such machinery, for example to monitor gearing and keep the engine at an optimum speed while maintaining a constant ground speed; or for ensuring a constant furrow depth when ploughing uneven ground. However, these systems tend to act as aids to the operator, improving the quality or efficiency of their work, rather than freeing them to perform other jobs.

Robots exist which can perform automated, repetitive tasks, and machines are available which humans operate to achieve more varied tasks, however, at present, the two are not seen in a single system. Fully independent robots are still not available either commercially or in research laboratories [Harsten, 1990].

It would be of great benefit to have independent robots which do not rely on constant human intervention, but are able to continue performing their job even if the goals of that job change slightly, or if the environment in which they are working changes. As well as the possible financial advantages such robots might bring to commercial operations, they could also be used for hazardous work such as in nuclear reactors [Mann et al., 1988] or in underwater situations [Herman and Albus, 1988], thus removing the need for humans to be placed at risk performing these dangerous jobs.

Humans are extremely complicated organisms, and if robots are ever to compete with them on any level, it is likely that they too will consist of a complex set of systems interacting with each other. Motion planning, therefore, is by no means the only requirement needed to allow the implementation of independent robots. Such a machine may require, for example, various tactile sensors, vision systems to observe the environment, and precise actuators to allow accurate movement and positioning of the robot. Nonetheless, path planning is an essential component without which a robot can only blunder round its environment randomly.

1.3 The Path Planning Problem

This section provides a high level discussion of some of the issues behind path planning. It firstly considers planning from an intuitive, human viewpoint in order to explain how the difficulty of the path planning problem varies, but without complicating the issue by considering the computerisation of path planning. Later, automated path planning is described showing how the problems of planning tasks of varying difficulty affect computerised planning.

1.3.1 Human Path Planning

The problems posed for humans when carrying out path planning vary considerably. For instance, the path planning involved in controlling a railway train poses a relatively simple problem. The train driver and signal controller must route the train so that the train travels safely from A to B without colliding with other trains sharing the same track. The path chosen may be optimised to make the journey as short as possible, or to minimise the energy used. However, there is no need for any planning to be performed to avoid collisions with objects in the train's environment such as bridges, verges or stations. The tracks are positioned to make such collisions impossible.

The path planning involved when driving a car is much more difficult. Besides the need for the relatively simple long term plan of deciding which roads to use to reach the destination, there is also the need for continuous, short term planning to negotiate bends, and to avoid driving off the road into a hedge.

Operating a mechanical digger presents a further level of difficulty, since the addition of a hydraulic arm allows the digger a greater freedom of movement. Because of this freedom, there are many more ways of moving the digging bucket from one position to another, but a skilled operator can easily cope with this sort of real-time path planning problem and would be able to adapt to different jobs, construction sites and mechanical vehicles relatively easily.

It is important to consider the rate at which plans must be generated. Different applications require planning to be performed within different time limits. For example a police driver involved in a high speed pursuit must have quicker reaction times than a driver of a road roller. In either case, if the rate at which planning must be performed exceeds the driver's capabilities, a collision will be inevitable.

1.3.2 Automated Path Planning

In automated path planning, the general requirement is the ability to move a vehicle between two points along a collision free course within a given environment. Two techniques often followed in achieving this goal are 'path planning' and 'obstacle avoidance' [Cameron, 1994].

Ideally the route should be such that the robot avoids collisions with other objects in the environment, whether they be stationary or moving (as in the case of other robots working in the same space). Certain robots, such as complex robot manipulators (robot arms), may even be able to collide with themselves — this too should be avoided.

Furthermore, the route which is computed should be optimised in that it should minimise some dependent variable, such as the distance covered or

the energy used, in executing the path. Typically there will be many possible paths between two points but only the most efficient is sought.

If there exists only one robot in an environment, with all obstacles remaining stationary, path planning alone may be sufficient for the robot to complete its task. The route which is pre-planned can be followed without any chance of collision. However, since the pre-planned route is generated on the basis of information available at a single instance, path planning alone may not be sufficient if the state of the workspace is continually changing as other objects or people move around within the environment. This need for obstacle avoidance also exists when the robot is working within an unknown, or partially known environment, when it is not possible to rely on pre-planned routes because of the limited information available to the planner.

In this case, the robot must have the ability to detect what is occurring in its immediate environment by means of some kind of sensors. When an unexpected obstacle is sensed, evasive action must be taken to avoid a collision with that obstacle. A new path can then be planned on the basis of the most up-to-date information available.

In order to develop fully independent robots, planning systems need to be developed which can generalise. Given previous knowledge of situations, humans are able to generalise to new, but similar situations. For a robot, it would be desirable to have a planner which does not have to return to first principles and generate the plan from scratch every time. Instead, the knowledge of previous experiences should be somehow stored and used to achieve future goals faster. There should not be a reliance on humans to instruct the robot as to precisely what actions to execute.

Each planner, from the hardware up, must be appropriate to its task. Planning in a simple, relatively uncluttered environment can be adequately achieved with a cheap processor, and a simple, exhaustive search algorithm. If the task requires planning in an application involving high resolution, a heav-

ily obstacle cluttered environment, and very high speed responses, then fast hardware, and advanced algorithms will be required. If a planner does not plan fast enough to avoid a collision, then it has failed.

1.4 Problem Solving with Computers

Path planning poses an important problem, a view borne out by the wealth of publications which offer different approaches to the path planning problem, as exemplified by the work of Barraquand and Latombe [1991], Werbos and Pang [1996], Rylatt et al. [1995], McLean and Cameron [1996] and Suh and Kang [1988]. The research into motion planning has given rise to a variety of techniques, each possessing their own merits and drawbacks. These are discussed later in Chapter 2. This section describes the conventional approach which is used to solve most problems in computing, path planning included. It recognises the problems of these approaches which particularly apply to difficult problems such as path planning.

The conventional approach to creating solutions to problems with computers has been to understand the problem to be solved, formulate an algorithm, and then implement this algorithm using one of many programming languages. The prevalence of computer technology demonstrates that this method has proved to be very successful in many cases, however this development method fails when not enough is known about the problem to manually produce an algorithm. A computer has no inherent intelligence, and as such cannot achieve anything without being told what to do by a programmer.

This problem is particularly prevalent when trying to artificially reproduce the skills exercised by humans. Humans, and other animals, move around and manipulate objects within their environment with apparent ease. Although algorithms for path-planning have been implemented, the problem of applying them is often intractable — it cannot be achieved in a useful time

for complex scenarios. No automated motion planning system can currently match the motion planning ability of humans [Hwang and Ahuja, 1992].

From personal experience we, as humans, know that we may make a conscious decision to perform an action such as moving a cup. However no such conscious effort is made in the planning of the individual movements which are required to complete the task. It is ironic that a task such as picking up a cup of coffee is so simple for humans and yet it is so difficult to make a computer carry out the same task. In contrast, a mathematical problem which many would consider complex to solve manually can be relatively easily programmed into a computer, providing that the algorithm is well known.

The lack of sufficiently competent artificially intelligent[1]systems suggests that our current knowledge of how humans process information in the brain, at either the physical or psychological level, is severely lacking. As our scientific knowledge increases, it may be that in the future we will fully understand how the brain generates solutions to path planning, and other problems. The fact remains that, at present, even the most powerful computers are not able to compete with simple mammals at these problems. This lack of understanding needs to be acknowledged and new methods adopted to address this problem. To this end, this dissertation describes the application of such methods drawn from the field of Evolutionary Computing (Section 1.6), and applies them to the difficult problem of path planning.

[1]The discussion of artificial intelligence can easily lead to philosophising as to what intelligence is and whether intelligence can be instilled in a computer, answers to which a series of dissertations would be hard pushed to address. The suggestion of artificial intelligence here in the very broadest sense implies a system which is not instructed exactly how to behave, but which indirectly, by the means of some algorithm, attempts to determine its own behaviour from a large set of possibilities in order to solve a problem.

1.5 Problems with Classical Path Planning

To date, most of the methods used to tackle path planning and obstacle avoidance can be considered to be classical approaches, relying on the conventional problem solving (Section 1.4) approach of applying, what may be, a limited understanding of the underlying principles in order to achieve solutions. They rely on applying, what may be, a limited understanding of the problem in classical computer problem solving.

Whilst there exist methods which provide us with perfect answers, the amount of computation required to provide the solutions may be very large. This is particularly true of multi-dimensional problems for robots with many degrees-of-freedom (dof). Degrees-of-freedom refers to the freedom of movement which a particular robot has. For example a translational robot which can move horizontally and vertically has two dof. A complex robot arm may have 6 dof. It is essentially the number of parameters required to specify the movement of the robot.

Although it has been stated that a computer is unintelligent, various methods are available, which may be referred to as Artificial Intelligence (AI). It is still necessary to program these techniques into a computer which consequently make the millions of components in the CPU exhibit apparently intelligent behaviour. The neurons in the brain are only intelligent because they have been connected in a certain way.

The evidence of the success of the intelligent human path planner suggests that the use of some intelligent, automated motion planning would be useful to investigate. With the adoption of intelligence, it can expected that very complex tasks will be performed by robots [Sheu and Xue, 1993, Chapter 1].

Traditional approaches to instilling apparently intelligent behaviour in a computer rely on explicitly supplied domain knowledge, from which inferences are made to determine the behaviour of a system given some input [Rich,

1983; Winston, 1992]. Whilst such approaches can provide excellent solutions to problems it can be a significant, if not impossible, task to provide all the necessary knowledge required to solve a complex problem [Angeline, 1994]. Assuming that this knowledge can be identified in the first place, the task of testing and updating this knowledge base is a considerable task, making some traditional AI approaches impractical [Angeline, 1994].

Because knowledge about a difficult problem, such as a complex path planning problem, is difficult to acquire, an appropriate alternative type of algorithm is one which is able to generate the required knowledge implicitly.

1.6 Evolutionary Computation

The advance in computing hardware has allowed a new branch of AI to come to the fore, namely Evolutionary Computation (EC). Techniques falling in the EC area are also referred to as Evolutionary Algorithms (EA). EC has become the standard umbrella term for a number of evolutionary driven techniques [Fogel, 1997]. Techniques encompassed by EC require either the speed and/or the large memory capacity offered by modern computers. The common feature of EC techniques is that, in various ways, they draw on Darwin's [1859] ideas of natural selection and survival of the fittest as the means with which to evolve data structures which represent potential solutions to problems. EC techniques are claimed to offer Emergent Intelligence (EI) [Angeline, 1994] which refers to the way in which they are able to create domain knowledge as part of their process, rather than requiring that it be supplied explicitly (as in traditional AI).

These techniques, including Genetic Algorithms (GA) [Holland, 1992; Holland, 1975] and more recently Genetic Programming (GP) [Koza, 1989; Koza, 1992; Banzhaf et al., 1998], have been shown to be capable of solving complex problems. Both GA and GP are iterative procedures which refine an initial population, or set, of randomly generated solutions, or individuals, which are

represented in a certain data structure. The use of 'population' and 'individuals' to refer to the set of solutions and a specific solution respectively is common in the EC field, and will, therefore be used in this dissertation.

In the case of the GA, the data structure is typically a fixed length binary string. A simple example is a problem involving the optimisation of a single integer parameter taking values in range [0..7]. The representation of the solution could be a 3-bit binary string which is capable of denoting eight discrete values. A population of randomly initialised 3-bit binary strings would be created and subjected to the automated evolutionary process to find the optimum value.

The GP stores its representation in a tree data structure, which generally makes it easier to work with solutions which are functions or programs operating on input variables from the problem domain. GP could be used, for example, to evolve a tree which represents a mathematical expression for a curve which approximates a set of training data. A cubic such as x^3+2x^2-x+5 could be represented in a binary tree form.

Both GA and GP assign a cost, or fitness value, to each solution in the initial population, and use these costs to select the better individuals for involvement in the next iteration of the evolutionary process. This selection is like natural selection in nature. The individuals in the subsequent generation of this simulated evolution are created either by directly copying them, by making a composite of two individuals, or by making a mutated copy of a selected individual. The process of evaluation, selection and insertion is then repeated until some termination criteria are reached — a solution is found, or a timeout has been reached. This is only a very brief description, and these evolutionary paradigms are explained in greater detail in Chapter 3.

GA and GP have been successfully applied to fields as diverse as Electrical Circuit Design [Bennett III et al., 1997], Control [Dracopoulos, 1997a], Medical Diagnosis [Kent, 1996] and Power Distribution Maintenance [Langdon,

1996]. It is also understood that there is interest in GA and GP from prominent organisations such as the Ford Motor Company [Hampo, 1992], British Telecom [Winter et al., 1994] and the British Defence Evaluation and Research Agency (DERA).

GA and GP have both been used in robotics for planning and obstacle avoidance [Handley, 1993; Ahuactzin et al., 1992; Davidor, 1991; Rylatt et al., 1995; Reynolds, 1994; Xiao et al., 1997].

1.7 Research Aims

The ultimate aim of developing a fully independent robot is beyond the scope of the research carried out for this dissertation. This dissertation necessarily focuses on a more specific component of an independent robot — namely path planning. The underlying thesis which this dissertation addresses is:

> Do Genetic Algorithms and Genetic Programming have a role to play in the future development of robot path planning?

In looking at this question, contributions are made to a number of different research areas, some more related to robotics, and some more related to evolutionary computing. To this end the overall research effort has been broken down by focusing on a selection of research aims.

- Identify existing approaches to path planning, and determine the advantages and disadvantages which are inherent in these approaches. This is to gain knowledge which is useful in fulfilling subsequent aims. Take examples both from:
  - evolutionary computing
  - non-evolutionary computing (classical approaches)

- Identify methods for applying evolutionary approaches to path planning either by using them to directly generate plans, or by using them to support a traditional path planning technique.
- Evaluate the suitability of the application of GA and GP to path planning by implementing and executing exemplar software and drawing conclusions from the results.
- Suggest how the results of the experimentation provide an indication of how GA and GP might be utilised in future path planning applications.

1.8 Dissertation Outline

This chapter has defined path planning and why, despite advances in technology, there is still a need for improvement in automatic path planning techniques. Existing techniques are either too slow, too computationally intensive, or are not able to generalise. GA and GP have been proposed as a possible means of improving future path planning techniques because of their success in solving difficult problems for which solutions have not been found using conventional problem solving methods. A number of broad aims have been specified which will be addressed over the remaining chapters of this dissertation.

Chapter 2 describes the path planning problem in more detail. The idea of 'path planning', which is investigated in this thesis, is placed in context by explaining where it fits in with other planning methods used in robotics. An overview of existing techniques used in path planning is presented and the advantages, disadvantages and ability to generalise are noted. Observations are made as to how GP and GA could fit in with existing systems.

Chapter 3 concentrates on the Evolutionary Algorithms, GA and GP. The underlying mechanisms which drive GA and GP are described, that is, how natural evolution is captured in these techniques to solve problems such as path

planning. The chapter discusses the practical application of GA and GP, and describes some of the inevitable pitfalls which may be encountered. Methods are described to help avoid potential drawbacks, such as the identification of a suitable representation scheme for the problem, and the use of parallel computing to speed up large runs. Some example applications of GA and GP are presented, including a number of planning approaches. These are examined to identify the drawbacks present in existing approaches, and to determine any useful techniques which might contribute to the subsequent design and implementation of an evolutionary path planning system.

Chapter 4 describes the implementation of a path planning system which is based on GP. Rather than use GP to evolve a specific path as a solution to a specific initial/goal position problem, the aim in this chapter is to evolve a generalised planning rule for a given workspace. So in effect GP is being used to evolve a planner rather than a single plan. A number of different approaches are proposed and examined and results on the performance of the techniques are presented.

Chapter 5 addresses the problems of the planner described in Chapter 4 by proposing a technique which seeks to combine GA and an existing classical approach to path planning. GA and GP have been shown to be excellent at optimisation, therefore this approach applies GA to the optimisation of an existing classical approach — the potential field method.

Chapter 6 provides an evaluation of the evolutionary approaches to path planning which have been investigated in this research. In particular it considers some of the issues which are important when using a planner in a real robot environment rather than in a simulated research environment. As well as highlighting the achievements made in this research, it also suggests what measures might be required to produce a real-world planner based on GA and GP.

Chapter 7 concludes this thesis by summarising the work carried out, sug-

gesting how others might continue the research, and by highlighting the contributions which have been made in this dissertation.

2

Classical Path Planning

2.1 Introduction

There is a need to place the research contained in this thesis in context. It was proposed in Chapter 1 that the path planning problem prevalent in robotics could benefit from the use of Artificial Intelligence (AI) techniques — in particular AI techniques which exhibit Emergent Intelligence (EI).

Before trying to apply these techniques, it is important to clearly understand the nature of the planning problem which is the subject of this research. What path planning is has only really been hinted at in Chapter 1. This chapter expands on the definition, and in doing so scopes the problem more clearly. The nature of the path planning problem is looked at from both the human and computer points of view. The ultimate goal for those carrying out research in robotics is to create an automaton capable of analysing, interacting with, and solving general problems in the real world. The design and construction of such a robot requires resources of time and experience beyond that available for this thesis. It is, however, possible to make a contribution by researching an important component of an independent robot; that is, path planning. If a robot is unable to plan a route for itself it will be of little use.

Having clearly identified the path planning problem, an overview of classical approaches which have been used to solve it is given. This explanation describes the way in which problems involving robots and their environments

are represented in a computer. It looks at different kinds of vehicles (robots) and workspaces, explaining the problems involved in their representation. A number of different techniques for generating paths are explained, and their advantages and disadvantages are investigated with a view to using this knowledge in the design of new approaches incorporating AI.

2.2 General Robot Planning and Avoidance

This section describes some different types of planning which are involved when using robots to solve problems. By discounting areas which are not of direct interest to the thesis, the research problem becomes more clearly defined. A number of different sub-systems may be present in a complete robot control system [Latombe, 1991, Chapter 1]. However, within the scope of this research, not all can be considered here.

Figure 2.1 provides a basic breakdown of the different areas falling under the general umbrella of planning. These range from the planning of individual tasks to achieve some larger goal, to specific methods for determining the trajectories to be followed by a robot. For the sake of simplicity this diagram is hierarchical, although in reality the areas are complexly linked with many overlaps. All the areas in this diagram will be discussed during this chapter, however most attention will be paid to the global planning sub-tree.

2.2.1 Task Planning

Most tasks can be broken down into sub-tasks. Considering this the other way round: given a sufficient set of sub-tasks, they can be executed in an appropriate order to complete a larger task. In the simplest of terms, task planning is the ordering of sub-tasks to achieve a greater goal.

This type of planning focuses on the domain of the problem rather than on the robot itself. A simple example is the process of making a cup of tea. The

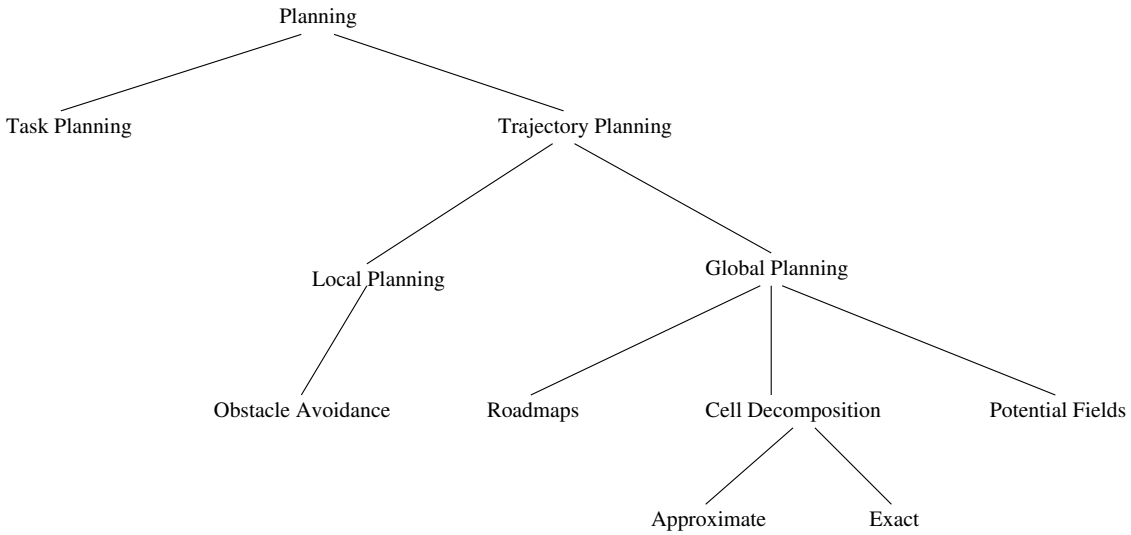


Figure 2.1: Different Types of Path Planning

problem involves a number of objects including tea bags, water, milk, kettle and cups. These objects may be in certain states, for instance the cup may be empty, or the kettle may be boiling. There are also certain actions to which the objects may be subject which will change the state of the objects. For instance the kettle may be switched on/off or poured. What is required in task planning is for a planner to find the correct set of actions which, when executed, will achieve the goal. A more formal description of task planning can be given as the process of identifying the sequence of actions required to transform from an initial state to a goal state. In the case of a cup of tea, the initial states will include kettle empty and tea in cupboard, and the eventual aim will include tea in cup.

One of the earliest planners was the General Problem Solver (GPS) which was developed by Newell and Simon [1963]. It used a technique called means-end-analysis — a hierarchical planning technique which works backwards from the goal to an initial state using a set of heuristic rules, some general purpose and some domain specific. Another example of an automated task planner is the STRIPS system developed by Fikes and Nilsson [1971]. The STRIPS approach searches for a sequence of operators which transform an initial model of the world to a goal model. The initial and goal models are

defined using well formed formulas (wffs). Similar wffs could be defined for the *cup of tea* problem described above. Using the wff notation, STRIPS is able to reason about the problem. It finds the action sequence which represents a plan to achieve the goal given the initial world state.

This type of work is by no means outdated. A more recent planner has been developed by Chapman [1987]. His planner draws on techniques used in previous planner work, but he showed that his planner used a set of techniques which were necessary and sufficient for solving the task planning problem. Nilsson's Teleo-Reactive (T-R) programs [Nilsson, 1994] provide a formalism which basically uses an ordered set of production rules to solve a problem. Each rule maps a condition to an action, and provided that the action of a rule i results in the condition of rule i+1, eventually all the rules will be executed and the goal will be met. Nilsson intends these production rule systems to be executed continuously, and allows features such as recursion and parameter binding. The actions are intended to correspond to electrical circuits (or programs) which could be present in a robot, and simple experiments in path planning for robots have been carried out. Since these programs are intended to be written by programmers, they are still not able to be independent. Nilsson himself hints at the possibility of automatically writing T-R programs, and indeed GP may provide the perfect method for their automatic generation in an application where limited knowledge of the domain is known.

Although it is important to perform automated task planning, it is not necessarily sufficient to give the robot or vehicle its independence. It is still necessary to plan the path in Euclidean space, rather than in an abstracted *task* space.

2.2.2 Path Planning

Path Planning or Robot Motion Planning is a fundamental problem in robotics, and has been the subject of research for many people [Glasius et al., 1995; Bar-

raquand and Latombe, 1991; McLean and Cameron, 1996]. It is necessary to plan a route for a robot, if one exists, between two points. The route should avoid obstacles in the environment, and is usually optimised in some way, for example with respect to length, or energy used.

This problem is a very general one. It applies equally to a theoretical point based, translational robot; to a vehicular car-like robot; or to a complicated, many degree-of-freedom (dof) robot arm. As the number of dof increase, the problem of planning a collision free path become very much more complicated, not least because a many dof robot arm may be able to collide with itself.

In general all the problems involved in planning deal with causal actions which have corresponding effects on the state of the world. Tasks or actions in the problem domain, such as *pour kettle* can be decomposed into robot level actions of applying torques to different link motors. There are two underlying problems which must be tackled, whether working in the problem domain or the robot domain:

- Predicting the effect (outcome) given the cause (actions)
- Determining the actions which will cause a given outcome

These are known as the forward and inverse problems and are described in more detail below.

2.2.3 Forward Problem

It is necessary when dealing in robotics to be able to predict the effect of a particular action or control signal. Assuming the robot can be modelled as a function, the set of control signals or actions represents the domain of the function, and the set of resulting changes to the state of the robot and its environment represents the range of the function. The forward problem is to

find a function, or a model of a robot, or the vehicle which is being considered. When driving a car we have some kind of model of the car, which tells us, for example that turning the steering wheel left will move the car to the left or that depressing the brake will decelerate the car. It is very difficult to undertake planning without knowing the resulting effect of specific actions.

2.2.4 Inverse Problem

The inverse problem is somewhat more difficult. It can be described fairly simply as the inverse of the forward problem. In this case a function is required which maps states back to the actions necessary to achieve them. It involves searching for the actions required to achieve a given goal. Continuing with the analogy of driving a car, consider the problem of parallel parking — reversing a car into a space between two other vehicles. In this case we know the target position and the current position, and we also know the result of any number of driver actions on the movement of the car.

Motion planning is an inverse problem rather than a forward problem. The goal is known, and what is required is the method of achieving the goal. The forward problem is still relevant if software is used to simulate a robot of some sort. A simulation must be able to mimic the corresponding effects of various signals to the simulated robot.

2.2.5 Obstacle Avoidance

Once a suitable path is determined by the robot by, for instance, applying power to wheels, or torques to joints. While the robot is traversing the predetermined trajectory, it may determine, through the use of external sensors, that the path previously planned is no longer valid as an obstacle now blocks the path. This may be due to limited knowledge of the environment, or because the obstacle is itself mobile and has moved into a new position since

the path was planned. Such occurrences require some form of obstacle avoidance (OA), to stop the robot colliding and to prevent accidents or damage to the robot. OA [Brunn, 1996] is clearly a very important component in a planning system but, like task planning, it is not within the scope of this thesis to consider this are in any greater detail.

The processes of task planning, motion planning and obstacle avoidance described so far are not necessarily executed sequentially one after another. The immersion of a robot in a dynamic environment is enabled by a system of interlinked components forming a composite robotic planning system. It is easy to imagine the complexity of a situation in which sub-tasks are planned, paths are planned to allow for the execution of those sub-tasks, a collision is avoided which requires for re-planning of the path, by which time the original task is no longer required so a new task is planned and so on.

2.3 Robots and Vehicles

There are a great variety of different types of robots, and their complexity varies enormously as the range of movements which they can perform increases. A robot can be described by a set of control variables, the values of which fall into a specific range. For example a car may have variables for its speed, and the angle of its steering wheels, whereas a robot arm may have variables describing the angles of each of its joints. The simplest robot (Figure 2.2a) is one which can translate horizontally or vertically like a rook on a chess board. It may take up an area in its environment, but usually for planning purposes it can be considered as no more than a single point on a sheet of paper.

A more complicated robot is one which is able to change the direction in which it is facing. One example is a car-like vehicle (Figure 2.2b(i)) which can propel itself forwards and backwards and change its direction via a set of steering wheels. Another is a robot which can change its orientation by moving its

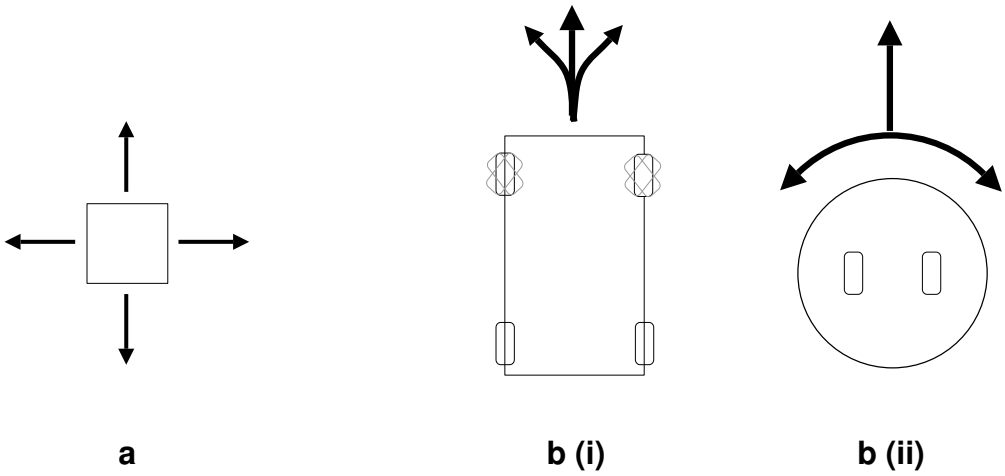


Figure 2.2: Examples of simple robots

driving wheels in opposite directions, an example of which is shown in (Figure 2.2b(ii)).

An important difference between robots b(i) and b(ii) is that b(i) can only change its direction by changing its position. This robot is said to exhibit non-holonomic movement [Latombe, 1991, Chapter 9]. The problem of controlling b(i) is more difficult than b(ii) as it is more difficult for the vehicle to move from one position to an adjacent position.

For instance, assume that it is necessary for each of these vehicles to reposition itself so that instead of facing north, it faces west, but still remain centred on the same point. For b(ii) the problem is relatively simple — the right hand wheel must be powered forward and the left backwards such that the robot rotates by 90°. The change in direction is achieved without any translational movement. It is more difficult to achieve this with b(i) because it must move away from the required point in order to change its orientation and then return to the point to complete the task. While doing this, it must avoid any obstacles in the vicinity; a problem not encountered by the other robots.

At the most difficult end of the scale are robot manipulators. These robot arms have a very wide range of movements, and subsequently have a large

state space. An example is shown in Figure 2.3 which is taken from a screen shot of the Simderella robot simulator package [van der Smagt, 1994].

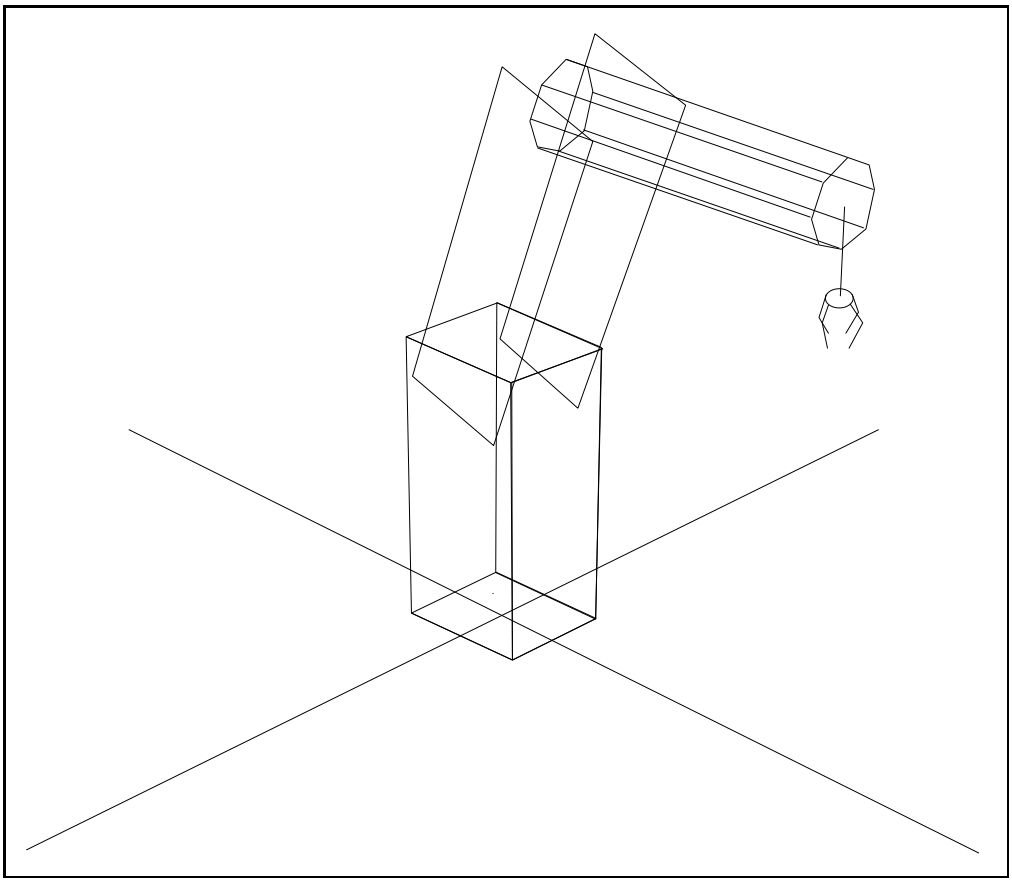


Figure 2.3: Example of robot manipulator

A robot manipulator comprises a number of links which are analogous to human limbs such as forearm or upper-arm. The links are connected by joints such as an elbow or shoulder joint. It has been shown by Reuleaux [1876] that there are just six different joint types: revolute, planar, cylindrical, prismatic, spherical and screw. In robotic manipulators, only two are commonly found: revolute and prismatic. Rotary joints are like human elbow or knee joints where the adjoining ends of two connected links do not move in relation to each other, only the angle between them changes. Prismatic joints allow a sliding translational movement of one joint relative to the other, but the relative orientation remains the same. Details of the others can be found in robotics textbooks such as Fu et al. [1987] or Selig [1992].

2.3.1 Kinematics

Kinematics is, in essence, a term for describing the forward problem in robotics. It is not concerned with the forces acting on the robot, but in the position of joints and angles of wheels. For simpler, vehicular robots as shown in Figure 2.2, kinematics of a simple sort are present, but kinematics tend to require greater consideration when working with higher degrees of freedom manipulators.

Although a robot manipulator may consist of a number of links and joints, it is the free end of the final link that is the point of interest. It is at this *end-effector* that work will be carried out, for example by a welding tool or gripping device. It must, therefore, be accurately positioned. The problem of finding the set of joint angles to position the end-effector in the desired position is the inverse kinematics problem (see 2.2.4).

The position of an arm could be described in any number of ways, but a uniform method of representing these types of manipulators has been developed by Denavit and Hartenberg [1955]. A manipulator consists of a number of links connected by joints. The end of each link may have an axis attached to it. The Denavit-Hartenberg representation defines a robot manipulator by describing the relative positions of successive link axes using a number of parameters. An example is shown in Table 2.1 for a PUMA robot manipulator. Θ_i and α_i define the joint angle between the ith and $i-1$th joints. The offset distances between successive joints are defined by a_i and d_i. Coordinate frame 0 is attached to the base of the robot. From these parameters, transformation matrices may be constructed which transform the base axes to that of the end-effector given specific values for the settings of the manipulator joints. This is a demonstration of the forward problem in robotics. Details of the application of the Denavit-Hartenberg representation may be found in robotics books such as Fu et al. [1987].

Joint i	Θ_i	α_i	a_i	d_i	Joint range
1	90	-90	0	0	-160 to 160
2	0	0	431.8 mm	149.09 mm	-45 to 45
3	90	90	-20.38 mm	0	-45 to 225
4	0	-90	0	433.07 mm	-110 to 170
5	0	90	0	0	-100 to 100
6	0	0	0	56.25 mm	-266 to 266

Table 2.1: Example of a Denavit-Hartenberg description of a PUMA manipulator

2.4 Workspaces

For a robot to function usefully, it must have some knowledge of its workspace. This workspace may be input to the system when a model based path planning approach [Sheu and Xue, 1993, Chapter2] is used, or a map must be built from sensors if a non-model based approach is used [Andre, 1994; Sheu and Xue, 1993, Chapter9]. Sensors may be used even when a model of the world is available for the purpose of *reactive* OA, rather than *pro-active* planning.

Like the robots which work in them, the nature of workspaces can vary greatly. It may be possible to use a discretised representation of a workspace stored in some kind of bitmap. An alternative is to define the workspace as a series of polygons. The representation used is dependent on the method used to perform the path planing.

A bitmap representation is easier to represent, possibly requiring little more than an array of the correct dimensions in which each cell contains a 0 for an empty cell or a 1 for a cell containing a part of an obstacle. Detecting intersections between obstacles and robots is simple, but a bitmap may be inefficient if the workspace is large, and of high dimensionality. The memory used to store the array contains a lot of redundant information between the obstacles, so in some cases a polygonal representation, where each obstacle is defined by a set of points, may be preferable. In this case the detection of collisions will be more complex, but this increased complexity may be more than justified in the reduction in storage space required. These issues are

found in other areas of computing where the storage of spatial information is required such as in computer graphics [Hearn and Baker, 1996] or in image processing [Gonzalez and Woods, 1993].

The dimensionality of a workspace is an important factor as this will greatly affect the space required to store the representation, and the complexity of planning within it. In many real world problems, the workspaces encountered by robots are 3-dimensional. It may, however, be sufficient, even for real world problems, to use a 2-dimensional representation of the workspace. The workspace for a complex robot-arm will be 3-dimensional, but for a car route planning problem, 2-dimensions are enough as the information in the third dimension is not important in this application because although a car is itself 3-dimensional, it cannot change its vertical position.

Whether a bitmap or a polygonal representation is used, the use of a computer will inevitably lead to some approximation of the workspace as computers operate in a discrete rather than continuous domain. The resolution of the representation must be higher if the application dictates it, however the result is that the memory required increases with the resolution.

2.5 Representing Robots within their Workspaces

Considering robots and workspaces independently is not sufficient for solving motion planning problems. What is important is the interactions of the robot with the workspace. Essentially the two state spaces must be combined into a single, multi-dimensional space of all the possible positions of the robot in all the possible locations in its environment which it may occupy. This can become quite large — consider a 6-dof manipulator in its 3-dimensional workspace. The state space for this common scenario is 9-dimensional. Add to this the range of values of each dimension at a resolution high enough to accurately model the problem and a very large set of states arise.

2.5.1 Configuration Space

An alternative way of considering this robot-environment state space has been proposed by Lozano-Pérez [1983] who defines the idea of configuration space. A configuration is a specific instance of a robot vehicle or manipulator within its environment. A number of different configurations may be possible even at a single position in the workspace, as a vehicle may be oriented in different directions, or a manipulator joint may be at any of a number of different angles.

The basic idea of Configuration Space or C-Space is to take a robot and its workspace, and reduce the robot to a single point, while expanding the obstacles in the workspace to take account of the shape of the robot. The original workspace is transformed into C-Obstacles and C-Free space. The appeal of this idea is that problems of path-planning or of positioning a robot or object within a workspace are reduced to problems involving a single point. It is easier, for example, to deal with the intersection of a single point with C-Space obstacles, rather than with intersections of obstacles and robot in a Cartesian workspace.

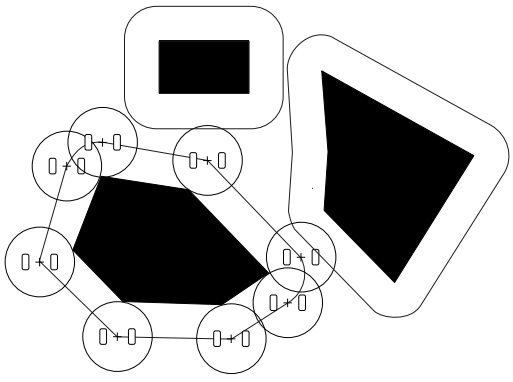


Figure 2.4: Creating C-Space obstacles

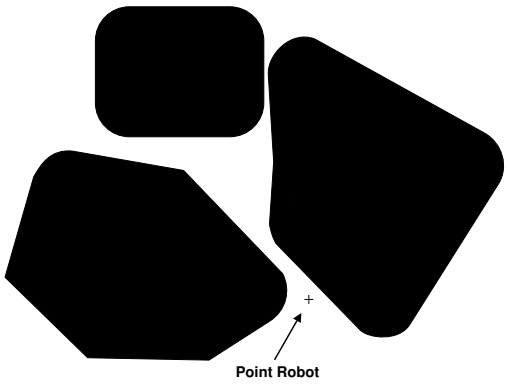


Figure 2.5: Working with C-Space obstacles and a point based robot

This process is demonstrated in Figure 2.4. In Figure 2.5 it can be seen how the problem is transformed to that of moving a single point, represented by

the cross-hairs, through the obstacles. This idea is not restricted to a 2-dimensional workspace, and can indeed be used for the many dimensional spaces required to solve robot manipulator problems.

The difficulty with this approach arises in the construction of the C-Space data structure containing obstacles. The mathematics can become quite complex [Latombe, 1991, Chapters 2,3], and therefore it must be considered whether the complexity involved in the construction of the data structure is justified by a reduction in complexity of the planning algorithms.

The ability to reduce a planning problem to a single point problem may be very useful in evolutionary path planning, as it may result in a reduction in the execution time of such planning. This issue is discussed in Chapter 3.

2.6 Classification of Planners

The complexity of planners varies with the particular application to which a robot is applied. This section examines various ways in which a planner may be classified.

2.6.1 Tracked and Untracked Systems

The simplest form of path planning can be achieved by restricting the vehicle to well defined paths or tracks which are guaranteed to avoid obstacles. These tracks can be modelled as graphs: arcs for the paths, and nodes at the intersections. Routes between two points on the graphs can be efficiently determined using standard graph search algorithms. This type of approach can be applied to robots which do actually operate on tracks, or which follow some virtual track such as a picking robot following a painted line on the floor of an automated warehouse. Path planning in off-the-shelf personal computer route planning packages can use these techniques, as roads can be considered

to be tracks. The fine level control of a vehicle on the road is of no concern to the route planner — it is assumed that the driver will keep it on the track or road.

Efficient algorithms to search graphs are well documented [Dreyfus, 1969; Even, 1979], a well known example being that of Dijkstra [1959] or the A* algorithm:

1. Form a one-element queue consisting of a zero-length path that contains only the root node.
2. Until the first path in the queue terminates at the goal node or the queue is empty,
   (a) Remove the first path from the queue; create new paths by extending the first path to all the neighbours of the terminal node.
   (b) Reject all new paths with loops.
   (c) If two or more paths reach a common node, delete all those paths except the one that reaches the common node with the minimum cost.
   (d) Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least cost paths in front.
3. If the goal node is found, announce success; otherwise, announce failure.

Whilst easy to manage, tracked systems do not provide sufficient freedom of movement for many applications. Once the vehicle is free to roam unrestricted by tracks, the path planning problem becomes significantly more complex, and computers cannot compete with humans.

2.6.2 Completeness

The completeness of a planner is a classification of how exact its planning ability is. Different degrees of completeness are:

Exact An algorithm is exact if it can always find either a solution to a problem, or prove that there is no solution. A typical problem with such an algorithm is the computational effort required. An exhaustive search will be exact, but for a complex problem its execution will take too long.

Resolution Complete This is a member of the exact group. The use of a computer algorithm means that often the problem must be discretised, and as such is no longer guaranteed to be exact. However, if, as the resolution approaches the continuous domain, the algorithm approaches an exact one, it is said to be resolution complete.

Probabilistically Complete These algorithms are not strictly exact. However where it can be shown that the probability of them finding a solution can be made to approach 1.0, they are said to be probabilistically complete. An example is simulated annealing.

Heuristic A heuristic search is guided by rules. This means that a solution will usually be found, and usually in a time faster than an exact method. However, if a solution is not found, this does not necessarily mean that one does not exist.

Exact planners which either find a solution to the planning problem, or prove that one does not exist [Hwang and Ahuja, 1992] are desirable in all planning applications. However the computational cost involved often associated with them means that they are not usable for complex problems. Instead, for practical application it is more likely that heuristically driven planners will be

used. They are useful in commercial applications where often the need is for fast planners and exactness is a secondary consideration. Exact algorithms are useful for theoretical analysis of algorithms and problems.

There is often an inherent inaccuracy involved in computerising an algorithm as the discrete rather than the continuous domain must be used. It is for this reason that the terms resolution completeness and probabilistic completeness are used. Planners which are resolution and probabilistically complete can be considered, for all intents and purposes, to be exact. If a workspace is broken down and represented as cells of a certain size, some approximisation is used — the representation of the real world can never be continuous. As the resolution is increased towards infinity, an algorithm may become more and more exact. In this case it is considered to be resolution complete. A heuristically driven method is not exact, but if the probability that it will find a solution can be made to approach 1.0, given infinite resources, then such a method is considered to be probabilistically complete.

2.6.3 Scope

The scope of a planner is described as local or global. A global planner considers the whole environment, planning a route from initial to goal position. A local planner is concerned only with the contents of its immediate surroundings. Global planners are the type which will enable the planning of a path from initial to goal position, as is required for this work.

Local planning is usually only used for planning in the very short term, as a means of obstacle avoidance. However, as will be described later in this chapter, it is possible to use local planning as part of a global planner.

2.7 Roadmap Methods

This section describes various roadmap methods, the first of a number of classical path planning approaches. Previously in this chapter, traditional

tracked systems were described, and it was noted that graph searching could be applied to such systems to find optimal paths along the tracks. It is possible to use the idea of tracks even when a system does not naturally use them, instead a virtual *roadmap* can be constructed for a problem. Roadmap methods basically involve constructing a graph of the workspace, in Cartesian or configuration space, which can be followed by a robot without it colliding with obstacles. This graph can then be searched, using the standard graph searching algorithms, previously described, to find the shortest path.

2.7.1 Visibility Graphs

A simple approach [Ó'Dúnlaing et al., 1983] is demonstrated in Figure 2.6 in which a simple problem is presented where there are two polygonal C-space obstacles. A number of nodes are created which correspond to the vertices of the C-space obstacles, and the start and goal position. A *visibility graph* is created whereby an arc connects a given node to all other visible nodes, that is without passing through an obstacle. The thick, grey line represents the shortest path in the visibility graph between the start and goal.

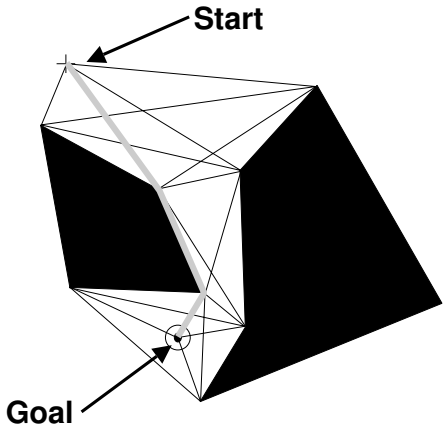


Figure 2.6: Example of Visibility Graph

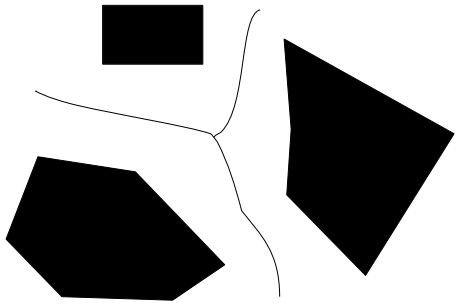


Figure 2.7: Example of Voronoi Diagram

An alternative to the visibility graph is the Voronoi diagram [Aurenhammer, 1991] which is a graph which follows a line equidistant from two or more ob-

stacles. A straight arc will exist between a pair of edges, or a pair of vertices, while a curved arc will exist between a (vertex, edge) pair. An example is shown in Figure 2.7. The Retraction Method of Ó'Dúnlaing et al. [1983] creates a Voronoi diagram which, like the visibility graph, can be searched using standard graph approaches.

The visibility graph creates paths which skirt the boundaries of obstacles, whilst Voronoi diagrams keep the robot as far from obstacles as possible. This may influence the decision to use one technique over another.

More generalised methods exist for roadmaps [Brooks, 1983],[Canny, 1988] which, for example, account for obstacles with curved edges as was seen in Figure 2.5.

In general, the problem becomes more difficult as the number of degrees of freedom increase. Planning for a typical robot manipulator may involve working in a six dimensional space due to the movements allowed by its elbow and wrist. The roadmap methods described above are not generally used with high dimensional configuration space, as they become inefficient, and other algorithms are available which produce better results [Latombe, 1991, Chapter 4].

2.7.2 Freeway Method

The Freeway Method [Brooks, 1983] approximates free space using a number of, so called, generalised cylinders. These cylinders are placed between pairs of 'facing' edges and are bounded by obstacles. The idea is best demonstrated with an example as shown in Figure 2.8. Each Freeway or cylinder has a spine. A freeway net can be constructed by using the spines as arcs, and the intersections of the spines as nodes. In this way a graph is constructed which has similar properties to the Voronoi diagram in that the robot is kept as far away as possible from obstacles in the workspace.

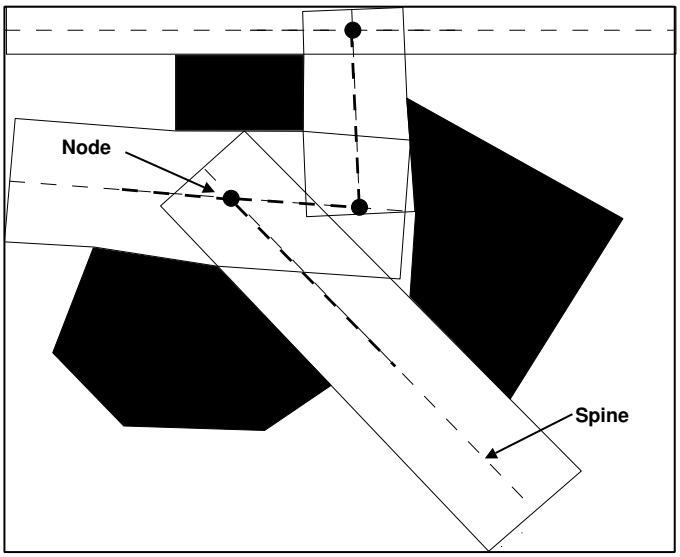


Figure 2.8: Example of the Freeway Method

The Freeway method is relatively fast, but incomplete, and therefore may not find a path even if it exists. This is particularly the case for a cluttered workspace. It is probably best to be content with this method as it is — fast but incomplete. Complicating it with variations will be likely to make it slower, but still incomplete [Latombe, 1991, Chapter 4].

2.7.3 Silhouette Method

Canny [1988; 1987] uses a method which generates a roadmap which borders all the obstacles in the workspace. It therefore reduces a multi-dimensional representation of a workspace into a 1-dimensional graph. The 'silhouette' is what would be seen if the workspace was made of perspex and the observer looked from above. The silhouette method builds the graph recursively by, for example, sweeping a 2-d plane through a 3-d C-Space to locate the edges of the obstacles, and then sweeping a 1-d line through the 2-d plane to identify unconnected sub-graphs, and joining them with extra arcs. For a space with more than three dimensions, the process remains the same except the n-dimensional C-space is swept using (n-1)-dimensional planes (hyper-planes), and recursively working down to the single dimensional line sweep. The idea

is demonstrated in Figure 2.9 for the 3-d problem — more dimensions than this are difficult to visualise.

To generate the path, the start and goal are treated as nodes and are connected to the rest of the graph, which can then be searched to find the shortest path.

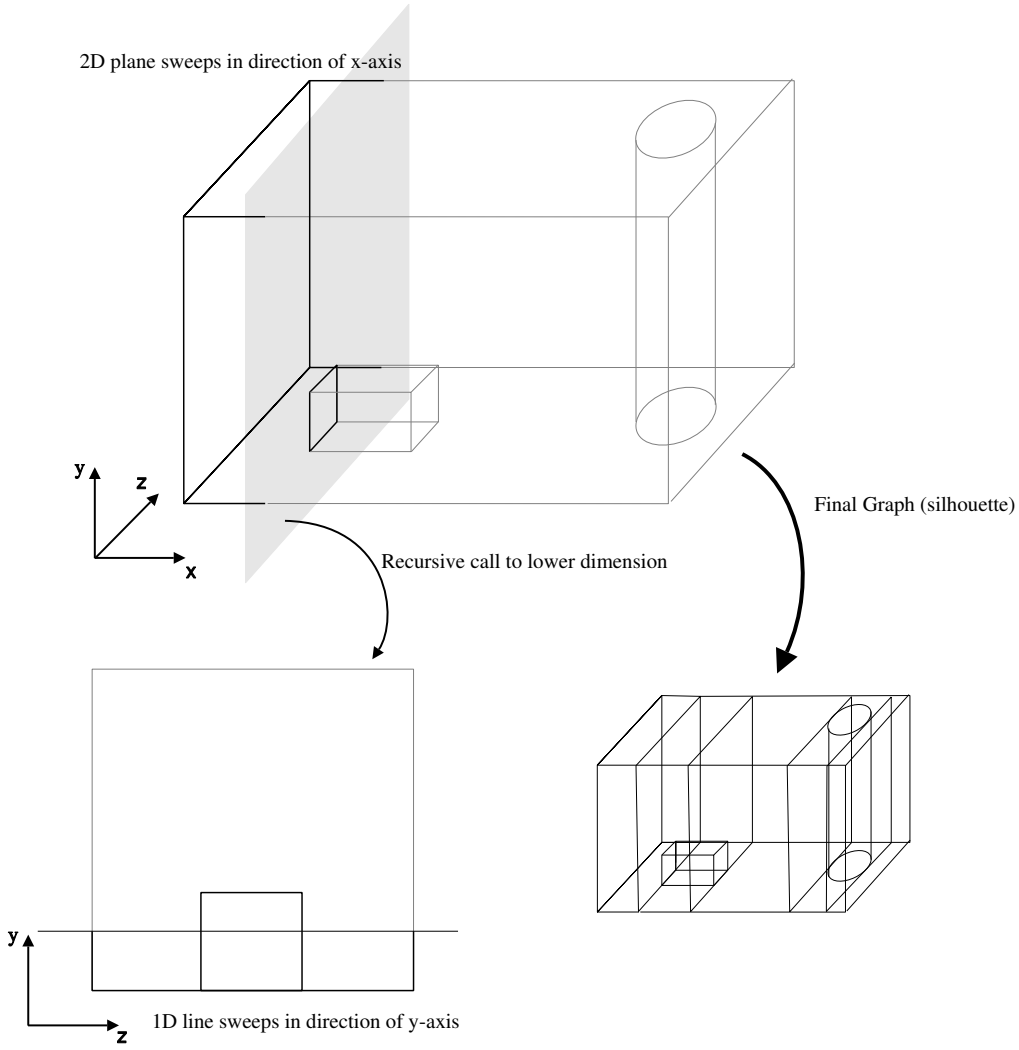


Figure 2.9: Example of the Silhouette Method

The Silhouette method tends to be used in the theoretical analysis of workspaces to find their complexity, rather than in practical path planning [Hwang and Ahuja, 1992].

2.8 Cell Decomposition

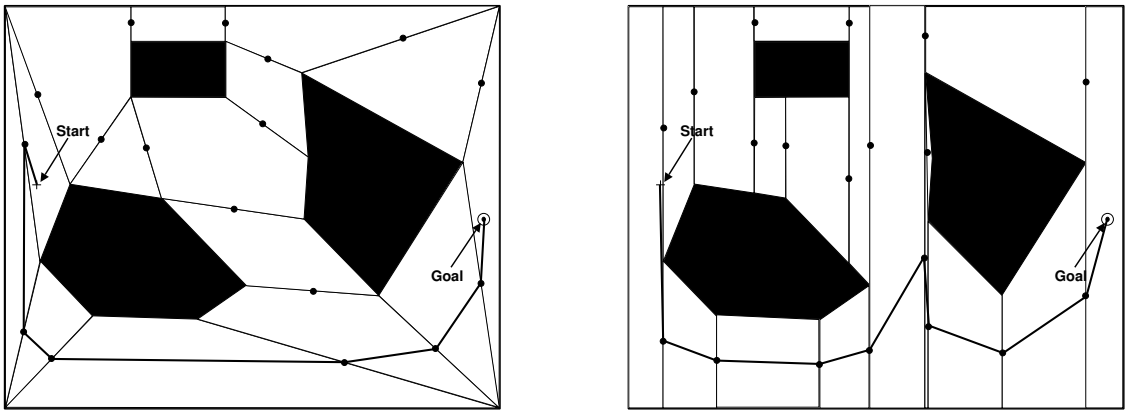
Another method used in classical path planning is cell decomposition which involves dividing the obstacle free space into a number of cells within which the robot can wander without colliding with obstacles. These cells do not overlap, in contrast with the cylinders of the freeway approach (Section 2.7.2). The initial and goal configurations will lie in free space in one of the cells. By finding a sequence of adjacent cells between the initial and goal configurations, a path can be found.

2.8.1 Exact Cell Decomposition

To exactly represent the free space using cells, a number of irregularly shaped cells must be placed in the workspace. These should be convex, since each point within a cell should be visible to all other points within the cell, so that in the event that the start and goal configurations are both with the same cell, creating a path is a simple case of connecting them with a straight line. Figure 2.10(i) shows the workspace divided into a number of convex polygons. Figure 2.10(ii) shows a representation which is also exact, but which is far simpler to compute, which uses trapezoid cells. The problem with the trapezoid approach is that the paths generated are unlikely to be optimal in Euclidean space, that is the distance travelled.

2.8.2 Approximate Cell Decomposition

Exact cell decomposition can involve time consuming algorithms. Approximate cell decomposition represents the free space using a hierarchical arrangement of regularly shaped cells. This means that the free space cannot be exactly represented, and therefore planning methods using approximate cell decomposition are not complete, although they are resolution complete, with the consequence that execution time is unbounded.



(i) Polygonal Decomposition (ii) Trapezoid Decomposition

Figure 2.10: Examples of the Exact Cell Decomposition of Free Space

A Grid of cells is placed over the workspace — this is simply a process of discretising the workspace. Each cell is then considered to be:

Empty When the cell falls entirely in free space.

Full When the cell falls entirely in obstacle space.

Mixed When the cell covers a mixture of obstacle and free space — that is it lies on the boundary of an obstacle.

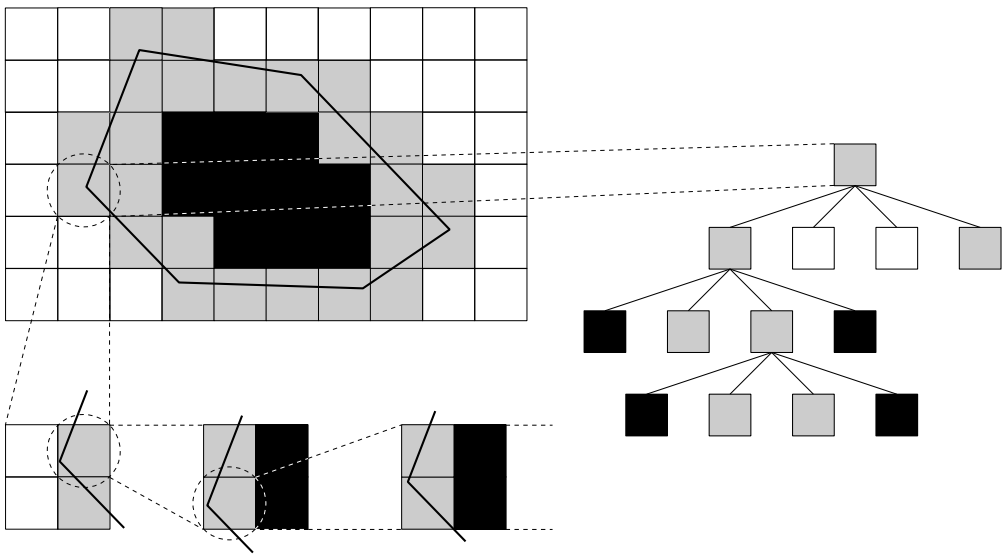


Figure 2.11: Approximate, Hierarchical Decomposition of Workspace

The simple case for path planning with this method is when a sequence of *empty* cells can be found between the initial and goal configurations. If this is not the case, then it may still be possible to find a path if a *mixed* channel exists — that is a sequence of adjacent cells consisting of *empty* and *mixed* cells. The area of free space covered by the *mixed* cells can be further decomposed at a higher resolution. In 2-d space, one cell would be decomposed into four sub-cells. This is known as a quad-tree representation. In 3-d space, an oct-tree representation is used with eight sub-cells, and in an n-dimensional space 2^n sub-cells would result. These sub-cells are all labelled as *empty*, *full* or *mixed* as before in the hope that an empty channel can now be found at the higher resolution. This hierarchical decomposition can be repeated ad-infinitum, but this theoretically means the algorithm would not finish in finite time (unbounded). In a practical application, it is more likely that a limit to the level of decomposition would be imposed. Figure 2.11 demonstrates the approach. A mixed square is decomposed at successively higher resolutions. As the resolution approaches the continuous domain (the cells are infinitely small), all the cells can be classified as either full or empty. This technique is not suitable for use with high dimensional spaces, as the space requirements grow exponentially because of the bitmap representation used.

2.9 Artificial Potential Fields

The basic idea behind the Artificial Potential Field (APF) is that planning can be divided into two components: one of pulling the robot to the goal, and the other of repelling it away from obstacles in the workspace. Assuming that the robot has a positive charge, a negative charge can be situated at the goal which results in a force which pulls the robot from its current position in the direction of the goal. Each obstacle in the workspace is also positively charged, the same as the robot. This means that each obstacle is surrounded by a field which repels the robot away. By combining the fields centred at

the goal, and each of the obstacles, a composite potential field results which should guide the robot towards the goal, but keeping it away from obstacles en-route. Having generated a field, the robot moves from its current position in the direction of the negated steepest gradient. More simply, it is like dropping a marble on the potential field surface and letting gravity pull it along the fall line to the goal.

The original use of the potential field idea in robotics is attributed to Khatib [1986; 1980]. The original use of potential fields was for obstacle avoidance rather than path planning with the primary aim being to avoid local obstacles and the aim of finding the goal was only a secondary consideration. In this respect it is considered a local rather than a global planner. Artificial potential fields have, however, been used for global planning as is discussed in this section.

The potential field operates on a single point. Most robots are not simple point based robots, so before applying potential fields some additional considerations need to be borne in mind. The potential fields can be calculated in C-Space (Section 2.5.1) whereby a more complex robot is reduced to a point whilst its environment is simultaneously 'grown'. To avoid the necessity of computing the C-Space for a problem, a number of points on a more complex robot can be selected as control points. Each of these control points is then independently subjected to its own potential field in the presence of the obstacles in the workspace.

2.9.1 Construction of Potential Fields

The construction of an artificial potential field (APF) in a computer is a fairly straightforward and fast procedure. The overall potential field U_{art} acting on the robot is given by [Khatib, 1986] :

$$U_{art} = U_{x_d}(x) + U_O(x)$$

Where $U_{x_d}(x)$ is the field attracting the robot to the goal and $U_O(x)$ is a sum of the repulsive potentials repelling the robot from the obstacles.

The attractive potential U_{x_d} is a parabolic well centred at the goal defined by:

$$U_{x_d}(x) = \frac{1}{2}\xi(x - x_d)^2$$

where x is the current position, x_d is the goal position, and ξ is a positive gain.

The repulsive potential fields U_O for each obstacle O are defined by the FIRAS (force inducing an artificial repulsion from the surface) function [Khatib, 1986] :

$$U_o(x) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho} - \frac{1}{\rho_0{}^2}) & if \rho \leq \rho_0 \\ 0 & if \rho > \rho_0 \end{cases}$$

where η is a positive gain, ρ is shortest distance to obstacle O and ρ_0 is the limit distance of the potential function, that is, the distance of influence of the potential field around the obstacle O.

Figure 2.12 shows what a potential field might look like.

2.9.2 Local Minima in Potential Fields

The problem with the potential field method is that of spurious local minima. What is ideally required is a field which has the lowest value at the goal and is monotonically increasing as the distance from the goal increases. There should be no local minima, that is points in the workspace for which the potential is lower than the immediately surrounding positions, other than at the goal itself.

Path planning with potential fields relies on following the line of the steepest gradient of the field from the initial position to the goal. If the position is in a trough, or on a flat area of potential, there is no downhill gradient which

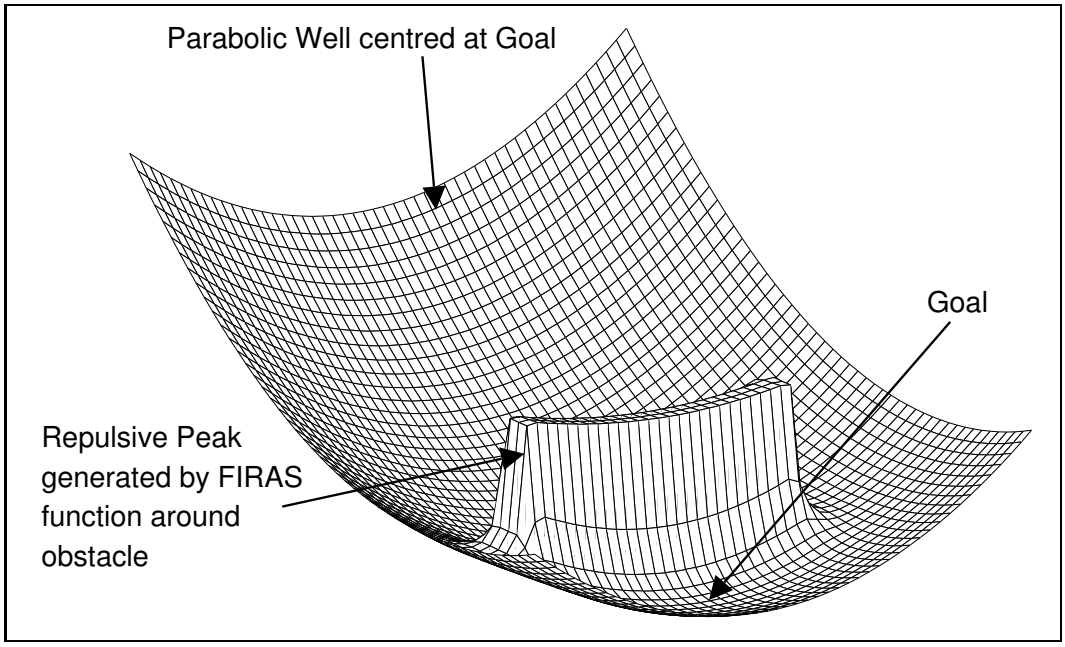


Figure 2.12: Example of a Potential Field

can be followed, and therefore the planner becomes 'stuck'. What is therefore required is firstly for the number of minima to be minimised, and secondly for their area of attraction to be minimised. A minimum which is very localised will not cause too many problems simply from a statistical point of view. If a local minimum is the minimum point of a large crater, then it is more likely that a path between two arbitrary points will pass through this crater and be sucked in, causing the planner to fail. As will be described later, it is also easier to circumvent minima with smaller attractive areas than those with large ones.

Unfortunately, from the planning point of view, it has been shown [Koditschek, 1987] that, in general, it is not possible to generate a potential field in the presence of obstacles with only a single minimum point at the goal. However the speed and ease with which potential fields can be generated means that potential guided planning is a useful method when coupled with methods to minimise or avoid local minima.

2.9.3 Avoiding Potential Field Problems

Randomized Path Planning

Given that it is inevitable that local minima will exist, methods have been developed for avoiding them. It is easy to detect when the planner is stuck in a local minimum because the robot will stop moving, but it will not have reached the goal as desired. The robot then needs to be given a bit of a push to move it out of the dip in the potential field, and then be allowed to carry on along the direction of steepest descent towards the goal as before. One technique used to escape a minimum is to apply Brownian motions, which are small random motions [Barraquand and Latombe, 1991]. This technique is implemented in the Randomized Path Planner developed by [Li et al., 1990].

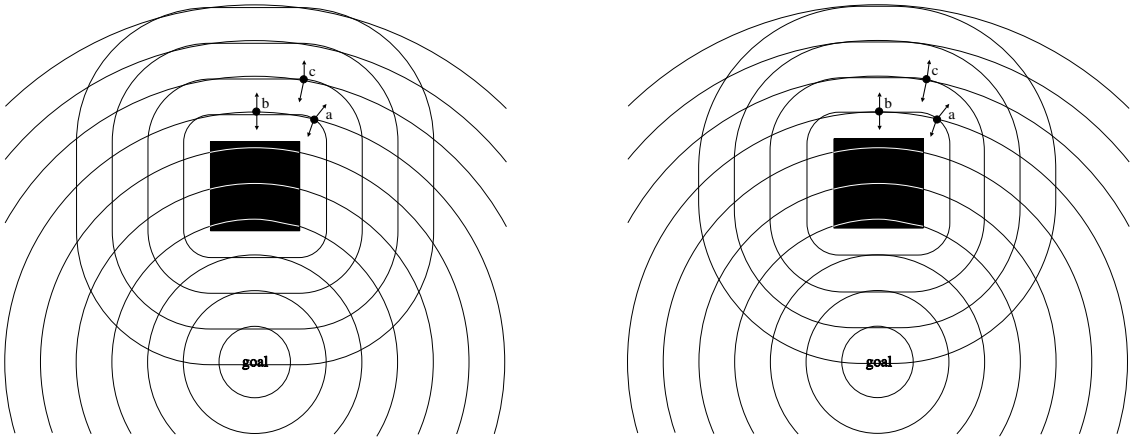
It can be seen that it is important that the wells are not too large as pointed out earlier. The larger the well, the larger the random force which is required to allow the robot to escape. If the well is very large, the force required to enable its escape will need to be so large that it will cause the robot to deviate sharply from its original course. The larger the well, the larger the deviation from the optimal route from initial to goal position which is required to avoid the well. The aim of the planned route is to avoid obstacles which exist in the real world, not to avoid spurious minima which are a side effect of using potential fields.

Virtual Springs

Another method used to avoid minima problems has been developed by McLean and Cameron [1996]. They perform planning for a redundant manipulator by applying APF planning to a number of control points on the robot. The difference between their method and conventional APF planning is that rather than modelling the robot as a set of rigid links, they model links as stiff linear springs, hence the name of the Virtual Spring Method. It has been found that

although errors are introduced using this method, because the model is not a true reflection of the real world, the number of local minima in the potential field is reduced because of the relaxation in the constraints on the system. The errors introduced are sufficiently small to allow them to be resolved later at the control system level.

Improving the Potential Field Function



(i) Rectangular Field (ii) Elliptical Field

Figure 2.13: Rectangular and Elliptical Potential Fields

Brownian motions and Virtual Springs build on the basic APF planning method. Another way of tackling the minima problem is to look at improving the potential field function itself. It has been proposed by Khosla and Volpe [1988], and previously by [Khatib, 1986] that elliptical potential functions could be used for generating the repulsive potentials surrounding obstacles. Figure 2.13(i) shows the equipotentials for a problem with a goal and a square obstacle using the FIRAS function which was described earlier. In Figure 2.13(ii) the obstacle is approximated by a field which is square at the boundary of the obstacle and becomes circular at infinity.

Three example points have been placed on each diagram. Each of the forces has two arrows attached indicating the forces due to the attractive potential field from the goal, and the repulsive potential field from the obstacle. In both

cases, a is subject to both a lateral force moving it away from the obstacle, and a downwards force pulling it towards the goal. The second point b hits a local minimum in both diagrams because it is on a line which is the line of symmetry of the obstacle and passes through the goal. This means that no lateral forces come into play to move the point round the obstacle. Point c successfully finds a path in 2.13(ii) but not in 2.13(i). In 2.13(i) the attractive potential causes some lateral movement, but because the repulsive equipotential at c has no lateral movement, the point will not be able to move round the obstacle. In 2.13(ii), the use of elliptical potentials means that there is some lateral force as a result of the repulsive field, and a local minimum will not occur.

In general, a local minimum will occur whenever the radius of curvature of a repulsive equipotential is greater than that of the attractive equipotential at a given point [Khosla and Volpe, 1988]. A flat wavefront has infinite radius, and so will always cause local minima.

The use of elliptical potentials for obstacles requires either that the obstacles are approximated by discs, or that elliptical potential functions are used for obstacles which are not actually elliptical. The first approach has the problem that it does not accurately model the workspace. There will be areas in the workspace which are made inaccessible by this approximation. The second approach has the problem that generalised elliptical functions can only be generated for certain convex shapes (trapezoids) in 2- and 3-dimensional workspaces [Khosla and Volpe, 1988].

Even without using alternative potential field functions, the number of minima can be varied according to the settings of the gain and distance of influence parameters. For example, a potential field which performs very badly can be achieved by setting the parameters to reflect the situation shown in Figure 2.14 where the goal is very near to an obstacle, and the gain or distance of influence is set too high. Where the goal is inside the distance of influence of the repulsive potential field of the obstacle, the global minimum

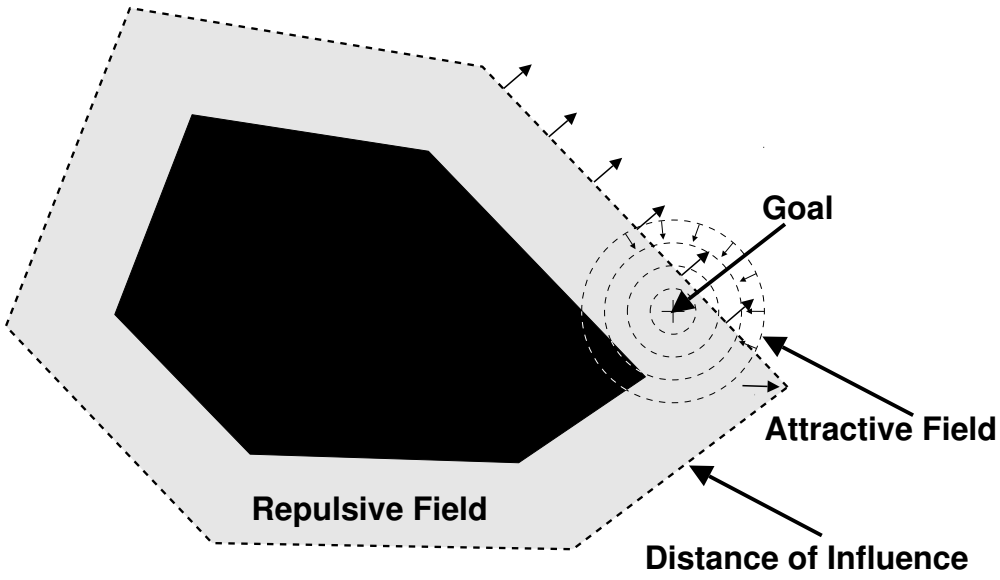


Figure 2.14: Example of a goal inside the distance of influence of an obstacle

cannot be at the goal because a minimum cannot exist inside the field of an obstacle. The correct setting of the controlling parameters is therefore a very useful way to optimise the performance of an APF planner.

2.10 Summary

This chapter has provided an overview of path planning which has allowed a variety of classical techniques to be assessed. It has been seen how they approach the path planning problem from perspectives directions, and details of the ways in which they succeed or fail at their job. This information is useful in the design, implementation and evaluation of improvements to existing path planning techniques.

Techniques which are related to path planning have been described, such as obstacle avoidance and task planning. It has been recognised that whilst they are important in their own right, and would inevitably form part of a complete robot solution, they are not to be the focus of this research.

Different types of robots have been described for which plans may be generated. It has been explained that they may operate in a variety of environments of varying dimension. It is important to realise that whilst the complexity of the path planning problem increases as the size, resolution and dimensionality of the environment increases, and as the number of degrees of freedom of the robot increase, the underlying path planning problem remains the same. Providing that this fact is acknowledged, it is valid to carry experimentation at a relatively simple level with a point based robot in 2-dimensions.

Having defined the area in which the research is to be conducted, and the problems which are pre-existing in the area, it is possible to begin investigating how to address these problems. The aim is to examine the possibility of a path planner for robots which is able to generalise, rather than plan from scratch on every occasion. The planner should also be practical, in that it should be able to produce timely plans and not plans which arrive too late to be of use in their application.

It is intended that in an attempt to improve on the current state of planners, the possibility of using intelligence in a computer should be examined. This view is pursued further in the next chapter by looking at traditional and more modern approaches to AI, namely EI, and how they have and could be applied to path planning.

3

Evolutionary Algorithms for Path Planning

3.1 Introduction

Having identified the domain for this dissertation in Chapter 2, this chapter introduces Evolutionary Algorithms as a means with which to address the path planning problem. This chapter begins by outlining how Evolutionary Computing (EC) differs from traditional Artificial Intelligence (AI), with which it is sometimes compared, and explains why its use is becoming more prevalent. A description is given of some EC techniques, focusing particularly on Genetic Algorithms (GA) and Genetic Programming (GP). Some prominent example applications of GA and GP are highlighted, which lend support to their worth, and the drawbacks of the techniques are noted with suggestions as to how these drawbacks can be minimised or removed. Finally, a review is given of the existing use of GA and GP specifically in path planning.

3.2 Artificial Intelligence and Machine Learning

Early in the history of computing, it was recognised that if computers were to solve complex problems, they would either have to be told explicitly what to do by human programmers in a very exact way, or developments would be

required to allow computers to solve problems without being told explicitly what to do.

Traditional AI as researched in the 1960s and 1970s focused on studying the problem domain, capturing pertinent expert knowledge, and building this knowledge into the system. Prominent examples are expert systems relying on large knowledge bases of rules such as XCON (eXpert Configurer) [McDermott, 1982] which provides configuration information for VAX computer systems ordered from DEC (Digital Equipment Corp.); and MYCIN [Shortliffe et al., 1973] which diagnoses certain antimicrobial infections and recommends drug treatment. Such systems have been very successful, and the techniques which they use continue to be used today in systems such as that implemented by Hammond and Davenport [1997] which is a CAD system for Dental Prosthesis design which automatically checks the correctness of the design. Each of these systems share the requirement that knowledge, perhaps in the form of a rule base, must be manually built into the system by a human. The drawback to these systems comes when research into the domain has not yet, or cannot ever reveal the information required to implement rules or develop an algorithm to solve a problem. Additionally there is a high financial and time cost associated with the maintenance of the knowledge in the system to ensure it is kept up-to-date [Hayman, 1997].

Research into systems which do not require explicit knowledge to be built into them has been carried out since the early days of computing. Motivated by the lure of AI or Machine Learning, Friedberg [1958], Schwefel [1975], Rechenberg [1973] and Holland [1975] have carried out research into areas which have developed into the present day techniques which fall under the synonymous umbrella terms of Evolutionary Algorithms (EA) or Evolutionary Computing (EC). These techniques allow computers to learn and to adapt solutions to problems without being told explicitly what to do. The term Emergent Intelligence (EI) is sometimes used to describe the way in which Evolutionary Algorithms create knowledge rather than having it built into them

[Angeline, 1994].

Modern EA techniques place a heavy burden on computer hardware [Banzhaf et al., 1998; Kent and Dracopoulos, 1996a]. Hence, early research into EA techniques was rather limited due to shortcomings in computer hardware. The continual improvement in hardware technology has allowed the use of EA techniques to gradually become more commonplace in computer science. The application of EA techniques seem to have particularly increased since the adoption of Genetic Algorithms (GA) and more recently with the advent of Genetic Programming (GP). As the computing resources which have become available during Hardware developments in the 1990s have resulted in considerable growth in the use of EA techniques as illustrated by the publication of a new journal for Evolutionary Computation [Fogel, 1997].

With this new found interest in EA, it has been possible to re-visit problems which have historically been tackled using more classical computer science approaches, and to use evolutionary methods to try to find improved solutions. Path planning, in particular, has been attempted in many ways, some of which have been explored in Chapter 2, however none of these offer a perfect solution. The approaches which have been taken have, in one way or another, sought to apply domain knowledge. This thesis proposes to investigate the suitability of applying Friedberg's [1958] alternative approach — that is allowing the machine to do something without being explicitly told how — in its application to path planning.

3.3 Evolutionary Computing

Nature presents us with an enormous number of examples of solutions to complex problems. Every creature is a solution to the problem of thriving within its environment. Each creature is a product of evolution.

Biological evolution operates on the DNA which is an enormous string of what is effectively a 4-bit code. This code is constructed from pairs of the nucleic

acid bases, adenine, guanine, cytosine and thymine. The representation of a creature in a coded form in DNA is called the genotype, whereas the expression of the DNA in the physical form of a creature with all its particular attributes is known as the phenotype [Johannsen, 1911]. This is analogous with a program written in a language such as C, and the resulting running application, which bear no physical resemblance to each other.

Evolution is an abstract concept; it cannot *understand* an algorithm for a fish and then implement it by expressing it using the language of DNA. Instead it efficiently searches all possible combinations of DNA (genotype) to find a representation which is expressed as a creature (phenotype) which is good at surviving under water. Evolution therefore learns how to produce a solution without being explicitly told how to do it, it is only guided by outside influences.

There are a number of techniques which can be classified as Evolutionary Algorithms. Each has a plethora of variants each tackling one issue or another, but they share a common feature. In some way they draw on the principles of evolution, as presented by Darwin [1859]. The techniques do not necessarily seek to copy biological evolution, but are driven particularly by Darwin's theory that 'survival of the fittest' is the driving force of evolution, and that a 'good' individual is more likely to reproduce, and consequently pass the genetic information, which makes it 'good', onto subsequent generations.

It is not the intention here to describe all EA techniques in detail. GA, GP and related EA techniques are covered in texts including: Fogel et al. [1966], Fogel [1995], Banzhaf et al. [1998], Koza [1992], Holland [1992] and Goldberg [1989].

3.4 Genetic Programming: Evolving Computer Programs

Although a number of people have worked in areas which eventually contributed to GP, the field became more popular in the early 90's with the publi-

cation of Koza's book [Koza, 1992] which followed earlier work on what were called Hierarchical Genetic Algorithms [Koza, 1989]. This indication of a relationship between GA and GP is no coincidence — indeed they are very closely linked. This section describes the Genetic Programming technique with a view to its application to the motion planning problem. It is important to bear in mind that many of the techniques, and considerations given in this section similarly apply to Genetic Algorithms. Rather than considering GA and GP as two distinct techniques, in this dissertation GA is seen to be a subset of GP because of the amount of common ground which they share. The differences between the techniques are discussed in Section 3.8.

3.5 The Simulation of Evolution

3.5.1 Initialisation of the Population

Initialisation is the first stage of the iterative GP process, as represented in the flowchart in Figure 3.1. An initial population of individuals (programs) is randomly generated at the start of the run. The programs are written in a prespecified language consisting of functions and terminal variables. If desired, this initial population could be interspersed with non-randomly generated individuals to give the GP process a head start. This might be a previously known good solution to a problem which is the best-to-date solution available. Hopefully, GP should either use the components of these non-random solutions to contribute to a better solution, or discard them if they have nothing to offer. Care must be taken not to unduly pollute the population with such previous knowledge, as the GP search may be inadvertently directed away from better solutions, and instead converge early on sub-optimal solutions because diversity in the population is lost.

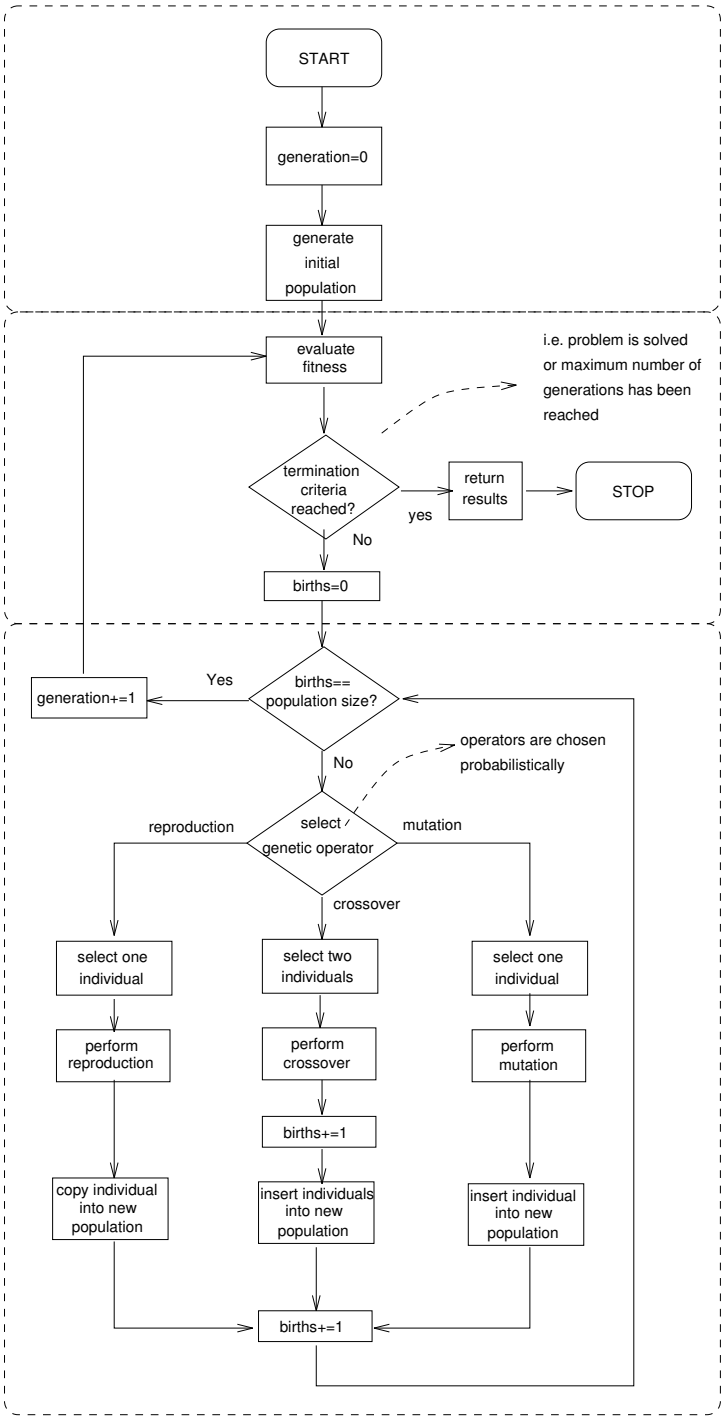


Figure 3.1: Flowchart of the GP process

3.5.2 Measuring the Population

Each individual program in the population is evaluated against a function (fitness function), to measure how well it performs against the problem it is addressing. The result of the evaluation is known as the fitness of the individual.

3.5.3 Natural Selection

The next step is the evolutionary stage where a new population of programs is created from the old population. Having measured the fitness, this information is used as the means of comparing the relative ability of programs to solve the problem at hand. During the evolutionary phase, those programs with a higher fitness are more likely to contribute either part, or all, of their structure to individuals in the new population. This process of fitness evaluation and evolution is repeated as the GP process efficiently searches for an optimum or near optimum solution to the problem at hand.

3.5.4 Breeding

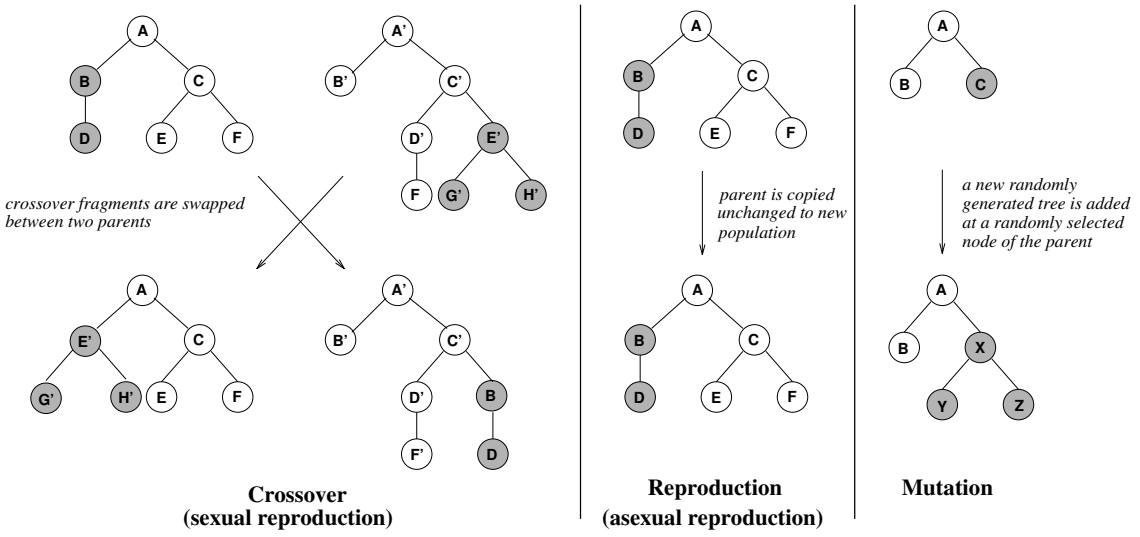


Figure 3.2: The main operators used in GP

The probablility of a solution arising in the first generation of a GP run is very unlikely, as this generation is created randomly. Solutions are found in GP by using operators which mirror the methods used in nature by which new offspring are generated, namely sexual and asexual reproduction. In nature reproduction involves the combination, or copying of sections of DNA. In GP, genetic operators are applied over successive generations, to the tree based structures which represent a solution to a problem. These operators are described below, and demonstrated in Figure 3.2.

Crossover

To produce new offspring, branches of individuals from the previous generation are combined using a GP operation called crossover which is akin to sexual reproduction. It is hoped that over successive generations, all the useful sub-components, or building blocks, which are initially spread throughout the population, will combine in a single individual which will offer a 'good' (near optimal), or even perfect solution.

Reproduction

To avoid the loss of good individuals from the population, and to improve the speed of convergence of GP, reproduction (asexual) is also used to copy some of the better individuals in their entirety to successive generations.

Mutation

Crossover and reproduction are the main operators used, typically to generate 90% and 10% of a new population respectively. Sometimes, it may be useful to introduce a mutation operator applied to individuals with a much lower probability (typically less than 1%) [Koza, 1992; Poli and Langdon, 1997]. The

mutation operation involves the random selection of an individual's subtree and the replacement of this subtree by another randomly created subtree. This alteration can be useful to prevent premature convergence to a solution which is sub-optimal, and can be seen as the addition of a small amount of random noise to the whole process.

3.6 Evolving a suitable Programming Language

GP evolves trees, but to solve a specific problem, the trees must be tailored by specifying a language or representation scheme which can express a solution to the problem. The scheme is specified in terms of functions which reside in the internal nodes of the tree, and take one or more arguments, and terminals which sit at the leaf nodes of the trees.

The definition of an appropriate language for the specific problem being solved is of paramount importance. If the representation scheme used for a problem is ill defined, then this alone may prevent a solution from ever being evolved. Sometimes it is very difficult to determine exactly what a representation scheme should contain. A necessary condition which must be met in order to solve a particular problem using GP, is that of sufficiency of the terminal and function sets. If a solution cannot be expressed in terms of the specified terminal and functions, then it is pointless to search for a solution using GP or any other technique. On the other hand, the search space grows exponentially with the number of terminals and functions, therefore their choice must be appropriate so as not to increase the search space unnecessarily.

A second condition which must usually be met is that of closure: the value of any terminal and the values returned by any function, must be processable by all other functions. That is, an evolved program must always be able to run. To this end, certain measures may need to be taken with some mathematical functions. A common example is division, which is undefined when dividing a number by 0. In this case a wrapper must be applied to the function, to

ensure that it does not simply generate an error, but instead returns a value, such as 0 or 1, which can be coped with by the other members of the function set. This is known as protected division [Koza, 1992]. An alternative to the closure condition, is to use strong typing [Montana, 1994]. This involves having functions and terminals which can return different types. Type checking mechanisms must be adopted during the creation of the initial generation and subsequent evolution of the population such that type inconsistencies are removed. In most cases however, a suitable closure can be defined for a problem without the complication and the extra time overhead of strong typing.

3.7 Other Issues for GP

Having provided an overview of the GP, this section looks deeper at some of the issues which needs to be considered when applying GP to a problem such as path planning.

3.7.1 How to measure a candidate solution

The choice of the fitness function used to measure the "goodness" of a candidate program is a key point for the successful application of GP. The fitness is the single means by which the GP process can choose which genetic material should be propagated from generation to generation. The GP process which relies on the feedback provided by the fitness value, resembles that of reinforcement learning techniques [Dracopoulos, 1997b; Sutton and Barto, 1998]. Individuals are punished (extinguished) if their fitness is low, while those with above average fitness are rewarded with their selection for further reproduction.

To obtain fitness values, each program in the population must be run or evaluated. The fitness value is what the GP run tries to optimise, therefore it must accurately capture the requirements of the problem. An ill-devised fitness

measure may inadvertently direct the GP search to the solution of a different problem. Examples of fitness measures might be an error value for a curve fitting application, the distance travelled in an obstacle avoidance application, or even a human supplied score as to the aesthetic appeal in a genetic art application. Several different types of fitness are described by Koza [1992, Chapter 6] but the important feature of fitness is that it provides a continuous scale of program performance to allow for comparison between programs, and the ability to identify progressive improvement in the population. A binary fitness measure which only measures perfect and imperfect will not allow the GP process to differentiate sufficiently between poor and successively better individuals.

Each program may be evaluated only once per generation, as in the Artificial Ant problem (Appendix C), or it may be evaluated many times against a training set of data, as in the Oral Cancer Diagnosis problem (Section A.2) — a classical example of supervised learning in which a set of input and desired outputs is provided.

The programs can be evaluated by allowing them to interact with the real world, for example in an embedded GP application to control a Khepera robot [Lee et al., 1997], evaluating programs by making the robot move around. Usually, however, programs are evaluated in a simulated environment. This is:

- flexible because scenarios can be easily changed
- cheap because expensive hardware can be avoided
- fast, as time can be speeded up inside a computer.

There is also a safety issue, as sometimes real world evaluation is infeasible as a poor genetic individual could cause damage to person or property.

3.7.2 Survival of the Fittest

The underlying principle of evolution is that of 'survival of the fittest', whereby 'good' genetic material perpetuates at the expense of the 'bad' genetic material. In nature this occurs through competition between members of a species to survive and therefore mate. In evolutionary computing, the selection of individuals to participate in the genetic operations is made according to an algorithm which considers the fitness of the individuals in the GP population.

The most popular method used for selection, known as **proportional fitness** or **roulette wheel selection** [Goldberg, 1989], involves calculating, for each individual, its proportion of the sum of the fitnesses of the whole population, such that the sum of all the individual fitnesses equals 1.0. The roulette wheel method is demonstrated in Figure 3.3. The diagram shows five individuals whose fitnesses have been normalised. The individual fitness of each individual is shown inside each segment, and the cumulative fitness of all individuals so far is marked outside the circle. The individual with the highest fitness of 0.35 covers the largest proportion of the circle and is most likely to be chosen, whilst the poorest individual with a fitness of only 0.05 is least likely to be chosen. When a random number is chosen, as indicated by the arrow marked 0.20, this is analogous to the spinning of a roulette wheel. The greater the area of the circle covered by an individual, the greater the likelihood of the random number falling in their segment, thus enforcing the principle of 'survival of the fittest'.

Another method often used is **tournament selection** [Koza, 1992]. It bears similarities with the type of competition occurring in nature where two animals will fight for the right to mate. Various rules exist for tournament selection, but in general two or more individuals are chosen randomly from the population, and their fitnesses are compared. The individual with the highest fitness wins. This type of selection does not require the evaluation of the fitness for all individuals in the population. It may be possible to evaluate

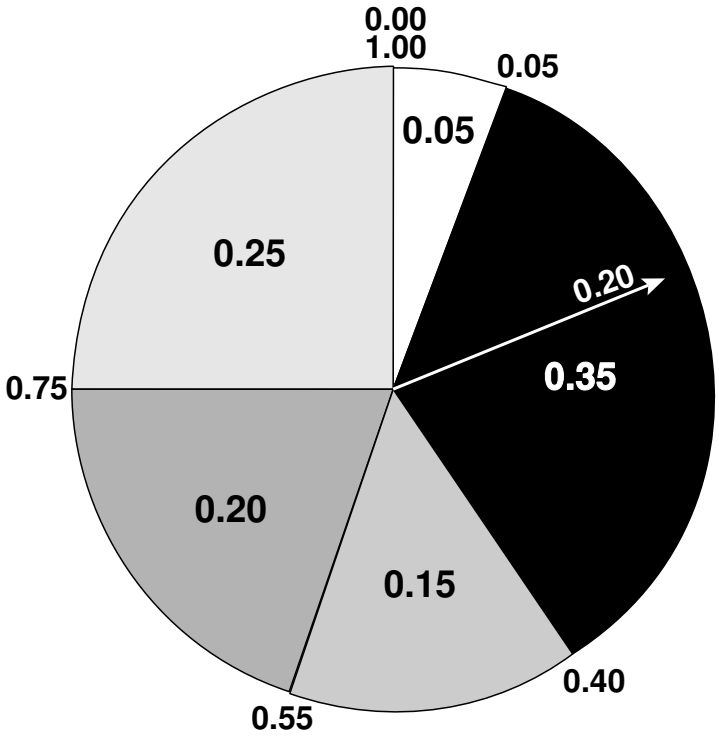


Figure 3.3: Demonstration of the Roulette Wheel Selection Method

only the fitness of an individual as and when it is chosen for a tournament. An alternative (**rank selection**) is to sort the individuals' fitnesses and rank them accordingly. During the tournament, the ranks can be compared rather than the fitnesses. The relative merits of different selection schemes are discussed by Blickle and Thiele [1995] in the context of GA, but this is equally applicable to GP.

3.7.3 The Population

To allow evolution to progress, sufficient diversity must exist in the population. In GP, a loss of diversity will result in the premature convergence of a run to a sub-optimal solution. The mean fitness of the population will become close if not equal to the fitness of the best individual. The population is "polluted" with duplicate genetic material such that new individuals, which differ sufficiently from the rest of the population, can no longer be generated.

The problem of loss of diversity and premature convergence can be caused by a number of factors. If there is insufficient diversity at the outset, then this is clearly going to cause rapid convergence. A simple way to ensure diversity at the beginning of the run, is to ensure that each randomly generated program is unique. A population which is too small for the problem being tackled is another way in which insufficient diversity may occur.

Assuming that the initial population is sufficiently varied, it may still be possible for diversity to be lost during the run. For example, the adoption of an elitist approach whereby the top n individuals are always copied to the next generation may speed up convergence, but may result in a sub-optimal solution caused by premature convergence. The use of asexual reproduction with a very high probability in combination with the proportional fitness selection method, may also cause loss of diversity because genetic material is propagated through the generations with little opportunity for new individuals to be created.

The evolutionary process described so far evolves a new population which replaces the current one. An alternative is to just replace one or two individuals in the current population and gradually evolve the population rather than to adopt a mass slaughter approach. This steady state GP (the equivalent of steady state GA [Syswerda, 1989; Whitley, 1989]) is less prone to premature convergence and it can improve the quality of solutions found by the GP run.

Another approach to maintaining diversity is to divide the population up into smaller, sub-populations referred to as demes. If communication of genetic material between demes is restricted to a small amount, the convergence of the global population is slowed, as diversity is maintained. This is because every few generations new genetic material is introduced into each deme as individuals are migrated between sub-populations. Although demes can be implemented in a single processing environment, they are very amenable and ideal for parallelisation as discussed in Section 3.9.

3.7.4 When to stop GP

The iterative GP/GA process previously shown in Figure 3.1 includes a point for termination of a run. Unless it is the intention for a GP or GA application to perpetually evolve to continually adapt to the changing environment, some termination criteria must be determined. The run may be terminated when the best individual in the population reaches some threshold level of performance. In some cases, a given run may not be capable of reaching this threshold, in which case another criteria, such as a maximum number of generations, may also be defined to prevent endless execution.

GP belongs to the class of probabilistic algorithms which give a different result every time they are run. Each run of GP is started from a random point according to the value of a random number seed. It is sometimes necessary to run the GP process a number of times, starting the search from different positions in the search space before a suitable solution is evolved.

Given the limited time and resources available for GP runs on a specific problem, it is generally more efficient to use fewer generations with larger population sizes, rather than the other way round. This is because populations converge much faster early in the run, so to maximise the improvement per generation, it is better to restart runs rather than continue them.

A common problem encountered in the case of multiple data sets is determining the correct stopping point of training. Whilst it may often be possible to find a perfect solution corresponding to the training data, this solution may not be able to generalise well on previously unseen data, a feature which is desirable for real machine intelligence. The problem is caused by over training (over-fitting of data) where the solution evolved learns a one-to-one mapping between the domain and range of the training data, rather than learning a general rule. It is, therefore, important to consider the generalisation error in addition to the performance of the solution against training data. An approach to achieve good generalisation is to divide the available data into three

sets: training, validation and test data. The training data is used to evaluate the genetic programs, and therefore to guide the genetic search. The fitness against this set should consistently improve. The performance of the best individuals is also periodically measured against the validation set. This should initially increase, but when successive generations result in a degradation in the performance against the validation set, the run should be stopped. In this way the second set is used just as a means of stopping the run at the best point. The performance of the best evolved individual can finally be evaluated on the third, unseen set of data after the termination of the run. Further details of this approach can be found in Dracopoulos [1997b, Chapter 4].

3.7.5 The Control Parameters of GP

Besides the function set, the terminal set, the fitness function and the genetic operators applied to the genetic programming process, a number of other control parameters must be determined, in order to apply the GP approach to a problem. First, associated with each genetic operation there is a probability p, which determines how often the genetic operation is applied to the current population.

A second control parameter which has to be determined is that of population size. A typical value of 500 is used many times, but depending on the complexity of the problem and the size of the search space, this value might need to be increased or decreased.

If left unchecked, the size of the data structures in GP could grow to an enormous size as evolution progressed. To stop this, limits are set on both the size of the initial, randomly generated programs, and the subsequent size to which they may evolve. If the use of a crossover or mutation operator with individuals would give rise to oversize offspring programs, the operation is cancelled, and a new operator/program combination is tried. The limits may

be set in terms of the maximum depth to which trees may grow, or the maximum number of nodes which trees may contain.

3.7.6 GP Problems and Solutions

Global and Local Minima

The aim of these optimisation methods is to find the global maximum or minimum; the highest or lowest cost (depending on the problem) across all possible solutions. A common problem encountered with all weak search methods is the avoidance of local optima. These local optima represent the best solutions to the problem in the immediate vicinity of the current point in the search space being investigated. They do not necessarily correspond to the best, global solution across the entire search space. Assuming the 3-dimensional landscape, these local optima are sometimes referred to as foothills, plateaux, and ridges. They are described below with reference to local maxima, but they are equally relevant for minimisation problems:

foothills If the search starts somewhere on a hill which does not have the highest peak in the search space, the search will be directed uphill to a high point, but this will only be a local maximum, not a global one. Having reached the local maximum, the search will not be able to find any adjacent, higher points and will terminate.

plateaux Although a search space may have peaks, these may be connected by flat areas. A hill-climbing method will not be able to follow the zero gradient on a plateau.

ridges A ridge of equal values in a search space may also cause the search to terminate on a local solution because, again, there is no uphill direction to be followed to achieve a better solution.

Computing Resources

The most obvious problems with GP are related to computer hardware. Given infinite power and memory, GP could solve any problem, although this is also true of a random search. As the complexity of the problems tackled increases, the computational and memory requirements increase very rapidly. This is because more complex problems typically require larger function and terminal sets and larger trees. This increases the size of the search space, which may mean that larger populations are required. Larger populations will obviously take longer to evaluate, and the time taken will increase still further if the simulator used in fitness evaluation is more complex. A larger population will require more memory. Even if the host machine has limited memory, the process should still be able to run if the operating system supports virtual memory. The drawback here is further increases in run time if a large amount of swapping takes place between real memory and hard disk.

Because of the limited resources available, good judgement must be applied in setting up the run without huge amounts of excess capacity which will increase the memory and computational requirements. Beyond that, attempts can be made to increase the amount of memory and processor power available. This task will fall partly to the hardware engineers, but improvements can be used by applying parallel processing techniques to GP as described in Section 3.9.

Parsimony

One claim sometimes made of GP is that it evolves solutions which can later be examined by humans. For instance it can be used to generate a function to approximate some data. The researcher can examine the function and may be able to gain some insight into the data. In reality, the evolved structures can be rather unwieldy, containing hundreds of nodes. Mathematical or logical

structures may be edited and simplified automatically either at the termination of the run, or during its progress. Another approach is to introduce the idea of 'parsimony' into the fitness. In this way the fitness is based not only on the problem being solved, but also on evolving a neat, compact solution to the problem.

A method to introduce parsimony in the solutions found is the extension of GP to automatically defined functions (ADF). According to this, GP is able to evolve subroutines (ADF) as part of the large general solution routine. The details of the incorporation of ADF to the standard GP is found in Koza [1994]. ADFs have also been shown to enable GP to solve problems more quickly than plain GP, or in some cases to solve problems which plain GP cannot.

The improvement of GP as a technique is still an area of very active research. It will be of great benefit to the GP field if the problems discussed above are addressed, such that convergence to optimum or near optimum solutions can be achieved in a smaller number of iterations.

3.8 Genetic Algorithms and their relation to GP

Genetic Algorithms and Genetic Programming are very similar. In the history of Evolutionary Algorithms, the GA came about before GP; major milestones being the book of Holland [1975] for GA well before the book of Koza [1992]. The main difference between the two techniques lies in the data structure which is evolved. In GA it is usually a string, and in GP it is a tree. The string may be of variable length, and each position in the string can take one of a number of states. It is usual to use a fixed length binary string. In some respects, the GA can be considered to be a subset of GP, as a string can be easily represented in a tree with each string position being represented by the leaf of a tree whose internal nodes consist only of unconditional branches. The distinction is further blurred when considering traditional programs written by human programmers. They write programs in a sequential fashion, and

not as trees. A suitable GA with a large enough alphabet, and typing constraints could be used to evolve a program. Indeed, genetic approaches have been used very successfully to directly evolve machine level code which can be run immediately without interpretation [Nordin, 1994].

In practice, there is room for both techniques to co-exist. It is often easier to define the representation scheme for a GP application. The nodes of a GP tree contain symbols which have meaning to a programmer such as mathematical or logical operators. The leaves often contain objects which correspond to control variables in the problem domain. The programmer is able to think about the problem domain, in terms of the problem domain.

GA is best used for evolving parameters which can be used to optimise a system. Using GA to tackle a problem which is more amenable to GP may require a somewhat abstract representation of the solution. GP is able to directly evolve programs, whereas the typical GA string representation does not lend itself so naturally to evolving program, or algorithm type solutions. A simple example is the artificial ant problem, described in Appendix C, which can be solved directly by evolving a simple program using GP, or more abstractly by using a GA to evolve the specification for a Finite State Machine [Koza, 1992].

For the purposes of the research described in this dissertation, it is considered that GA and GP can treated as one. They rely on the same underlying principles to perform efficient searches of large spaces for solutions to problems. The difference in the data structures which they manipulate mean that each is amenable to slightly different problems: GA to problems which require the optimisation of a set of values, and GP to a problem which naturally requires a rule, program or expression as a solution to the problem.

3.9 Speeding Up Genetic Programming

3.9.1 The Problem

Whilst GP has been shown to be able to solve difficult problems, an unfortunate characteristic is the considerable computation which is sometimes required by the process. The maintenance of the population, and the simulation of the evolutionary process is usually a very small proportion of the total run time. The area where most execution time is spent is in the evaluation of the fitness of the individuals in the population. Execution time can be particularly long when a complex simulation is required. This must, of course be carried out for every individual in the population. Difficult problems, with computationally expensive simulations inevitably demand larger populations, thus the problem is exacerbated. Whilst the ever increasing performance of modern CPUs will make more and more problems amenable to GP, in the immediate term, parallelisation of the technique offers the most effective means by which GP can be made to produce more timely results.

The idea of parallelising GP is by no means new. The Evolutionary Computing community have recognised some time ago that parallel computing could be used to improve performance in their field [Shonkwiler, 1993; Stender, 1993; Cantú-Paz, 1995; Jones and Valenzuela, 1995]. It is a natural progression to apply similar techniques to GP [Juille and Pollack, 1995; Andre and Koza, 1996]. A good example of applying parallel GP to a complex problem is provided by Bennett III et al. [1997]. In this approach GP was used to evolve analogue electronic circuits. To evaluate the fitness of the circuits, a patched version of the freely available SPICE (Simulation program with Integrated Circuit Emphasis) simulator was used [Quarles et al., 1994]. The experimentation was run on a system consisting of 64, 80MHz PowerPC 601 processors the architecture of which is described in Andre and Koza [1996]. Although it is reported that this is an efficient means of parallelising GP, the equipment required is inaccessible to most GP researchers.

3.9.2 The Bulk Synchronous Parallel Approach

It was felt that parallel GP should be available for those researchers who had more usual computing resources at their disposal.

Most university researchers have access to a network of uni-processor machines which can communicate with each other via a local area network. It is desirable for all researches to have access to parallel GP, because of the improved performance which can be achieved, and it was felt that a network of computers might provide the means to develop a parallel GP system spread across a number of computers rather than within a single, very powerful machine.. The disadvantage of parallelisation using this approach was relatively slow communication afforded by a typical Ethernet network. The important question was whether a beneficial speedup could be achieved with such a system.

Rather than build the parallel GP system from scratch, a parallel library was used in the conversion of a custom-built C-based uniprocessing GP system to a parallel version. The Bulk Synchronous Parallel (BSP) model of parallelisation [Valiant, 1990] was chosen as the framework within which to develop the system. BSP is a simple, SIMD (Single Instruction Multiple Data) parallel model.

The choice to use BSP was made for a number of reasons. It is a simple parallel model which is easily implemented through a library such as that developed by Miller and Reed [1993]. The library contains only six operations as shown in Table 3.1. Source code is available for the library allowing it to be recompiled on a variety of platforms. It is flexible because a number of different communication media are supported, including shared memory which can be used with, for example, a relatively cheap multi-processing SUN workstation. It is scalable because it offers a solution on a range of machines from PC clones running Linux, to machines with Alpha processors, and even Crays. This offers the researcher the opportunity to develop an application

on cheap machines, and if necessary to perform final experimentation on a massively parallel supercomputer without having to re-write code.

`BSP_START`	start of the BSP program
`BSP_FINISH`	end of the BSP program
`BSP_SSTEP_START(n)`	start of superstep `n`
`BSP_SSTEP_END(n)`	end of superstep `n`
`BSP_STORE(to, from_data, to_data, length)`	store from local to remote processor
`BSP_FETCH(from, from_data, to_data, length)`	fetch from remote to local processor

Table 3.1: The Oxford BSP Library basic operations.

The emphasis for this parallel system was for the implementation on a network of workstations. A particularly important requirement for this was that communication should be kept to a minimum, as this was likely to be where a bottleneck would occur, because any parallel GP system would inevitably be competing with network traffic from other users.

In order to minimise communication a loosely coupled approach was adopted, whereby the individual processes in the BSP machine were able to proceed independently, with communication occurring periodically. It was noted in Section 3.7.3 that the use of demes, or sub-populations have been shown to be beneficial for maintaining genetic diversity in GP. It is logical to locate a sub-population or deme at each processing node. In this way each processing node can be considered to be an island. This island model has been previously adopted in the GA field [Gordon and Whitley, 1993] and in parallel GP [Koza and Andre, 1995]. The extreme case of loosely coupled system would involve the simultaneous execution of a number of differently seeded runs as completely independent processes [Harris and Buxton, 1996], but this does not take advantage of the use of demes.

The standard GP process is modified by the addition of a migration operator as shown in Figure 3.4.

The processing islands are arranged according to a topology, examples of which are shown in Figure 3.5. Every 10 generations, the top 10% of individuals from each island were migrated to neighbouring islands consequently

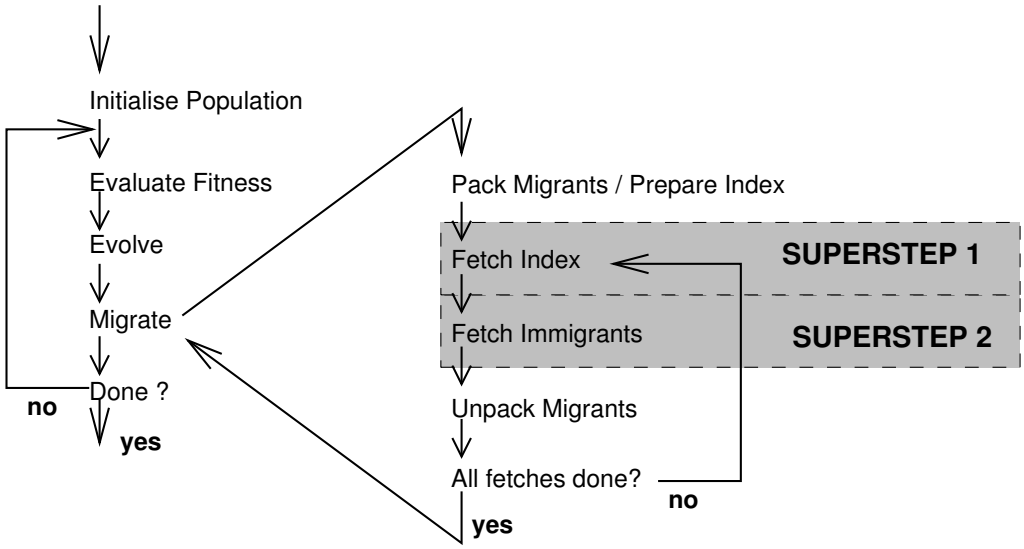


Figure 3.4: Parallel GP process

displacing the lowest performing individuals, thereby distributing the best genetic material throughout the global population.

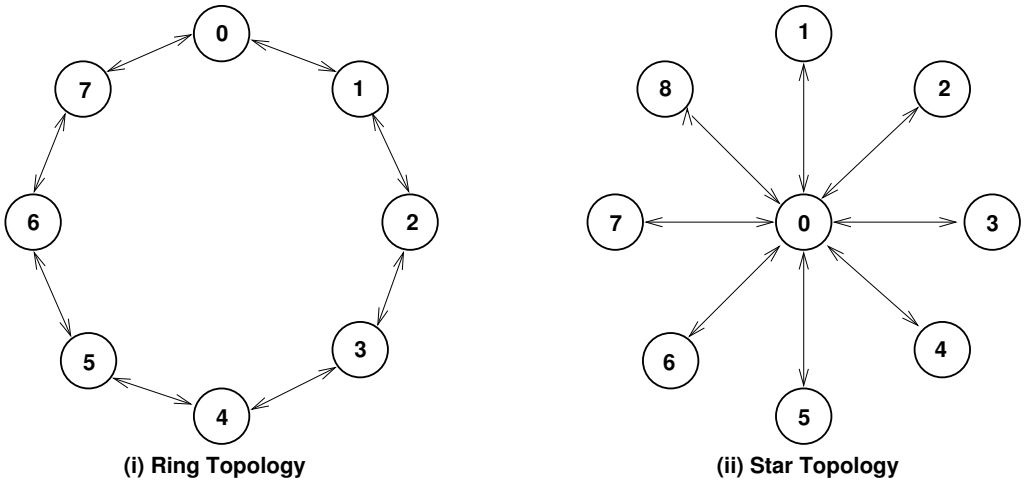


Figure 3.5: Topologies used with the island model GP implementation.

3.9.3 Results

The system which was developed was first tested using a well known problem in the GP field: The Artificial Ant Problem [Koza, 1992]. This is described in Appendix C.

The workstations used to run the software were a number of SUN SPARCstation 5 machines with 70MHz microSPARC-II processors, and running SOLARIS 2.4, each with 32Mb of RAM. These machines were connected on a common subnet with Ethernet cabling. Communication between processors was achieved using TCP/IP.

The two implementations were run for 50 generations, and the mean elapsed time was recorded. The results are shown in Table 3.2.

processors	elapsed time/s	
	ring	star
1	4980	5100
2	5280	5400
4	6180	6120
8	7020	6430

Table 3.2: Elapsed time for runs of 50 generations for parallel GP implementations

The recorded speedups are shown in Figure 3.6.

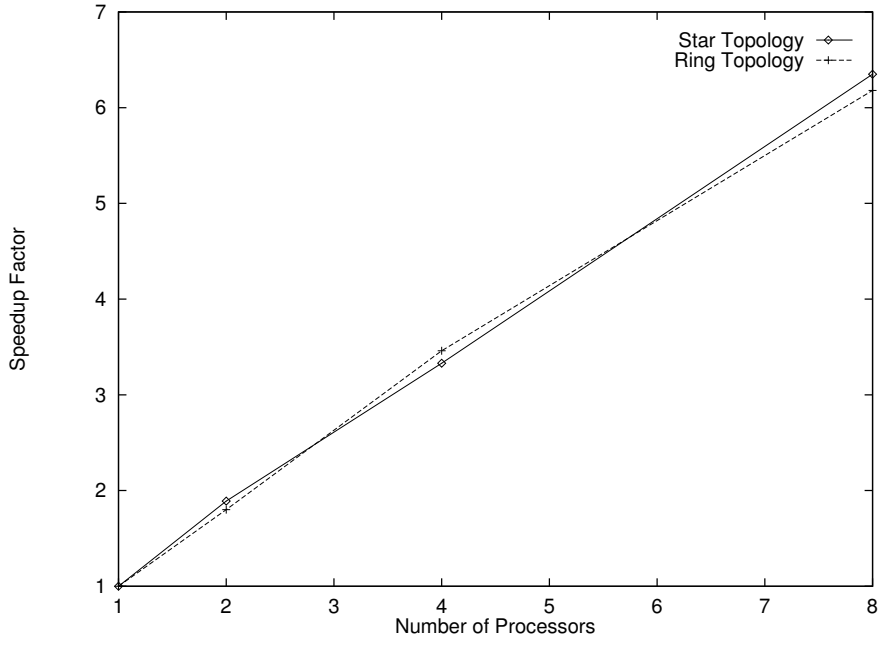


Figure 3.6: Graph of actual speedup achieved

The results demonstrate that a significant speedup can be achieved using this simple approach to GP parallelisation. The size of this Artificial Ant

problem is not very large, and it can be expected that problems requiring more complex simulation during fitness evaluation can further benefit from this approach, because the delays resulting from communication will be smaller compared with the actual parallel execution time. By making use of demes or sub-populations, the diversity of the population is maintained during the run.

During the development of this parallel approach, attempts were also made to distribute the processing of a single GP population over several processors, using a master-slave approach. The run was controlled from the master node which directed slave nodes to perform fitness evaluations. Because of the communication overhead this approach was not found to produce the same significant speedups as did the island approach.

Although the size of the problem was not very large, it can be seen that a significant speedup of the GP run is achieved. The differences between the star and the ring topologies are negligible. As the size of the problem increases, it can be expected that further speedups will be seen as the delays resulting from communication will be smaller compared with the actual parallel execution time.

Although the system was initially tested on the Artificial Ant problem, because the system was a general GP problem solver, it can, and has, been used throughout the research for this dissertation. The results of this research into the BSP parallelisation of GP have been published in [Kent and Dracopoulos, 1997], [Kent and Dracopoulos, 1996a] and [Kent and Dracopoulos, 1996b].

3.10 EA Applications

The EA field has developed very rapidly over the past few years. For example since its birth in 1992 the number of entries in the GP Bibliography

[Langdon, 1999] has grown to over 1000 entries. Although much of this research is theoretical, many examples can be found of research into the application of GA and GP.

Applications have been found in medicine and biology. GP as an automated diagnostic tool is presented in Section A.2, but there are also problems in the more theoretical molecular biology where there is currently extensive research into the identification and classification of DNA and protein sequences. Traditional approaches to research in molecular biology are very labour intensive and the structures which are being researched are enormous. The human genome contains roughly 100,000 genes, each containing around 1,000 bases. Intelligent techniques such as GP are being adopted [Handley, 1994b; Handley, 1995; Koza and Andre, 1996] to support the traditional methods, and thus speed up research.

One of the most interesting areas of GP/GA application is in the evolution of electrical hardware. Koza et al. [1996] have used GP to evolve analogue electronic circuits. The technique uses component creating functions and connection modifying functions to generate the final circuit by operating on a simple embryonic initial circuit. The evolved circuits are evaluated on a patched version of the freely available SPICE (Simulation program with Integrated Circuit Emphasis) simulator [Quarles et al., 1994]. This is a very computationally expensive application which uses parallel computing as described in Section 3.9.

Over the past couple of years some very interesting work has been developed on the direct evolution of Hardware. This Evolvable Hardware (EHW) [Higuchi et al., 1997] field has arisen only due to the availability of the Field Programmable Gate Array (FPGA). This is a large array of cells which can be instructed to act as one of a number of logic gates. The device may be programmed as many times as required, thus allowing direct, hardware evaluation of GA/GP structures. One, or a mesh of FPGA devices may be connected

to a standard PC hosting the evolutionary process. Work in this area was pioneered by Thompson [1996a] and has more recently been adopted by other researchers [Koza et al., 1997; Liu et al., 1997]. The specialist hardware required undoubtedly presents a barrier to entry into this area, but the promise of faster GP is likely to give rise to a considerable increase in research in this area.

Successful applications have been developed which use hybrids of a number of Evolutionary techniques. For example Rizki and Tamburino [1998] combine GP, Evolutionary Programming,and GA in a pattern recognition application which classifies radar signatures. GA have also been hybridised with other machine learning technique, for example Dracopoulos and Jones [1997] combine neural networks and GA to control the attitude of a satellite. These successful hybridisations lend support to the idea that a successful path planning application could combine an evolutionary approach with another technique.

3.11 Existing Evolutionary Approaches for Planning

The evidence of the use of GP and GA in various applications provides a useful indicator of the worth of the techniques, but this thesis proposes that the use of such techniques is of benefit specifically in the field of path planning. They have been used extensively in various areas in robotics [Alander, 1996], and this section presents examples of previous work specifically in the area of planning, which will be built upon over the remainder of this dissertation.

3.11.1 GP Task Planning

[Handley, 1993; Handley, 1994a] used GP to generate plans for a robot. This is an example of task planning where a robot is expected to transform a world from one state to another state. The world in this case consists of three rooms containing objects which can be manipulated. The goal world is presented

to the planner as a single fitness case, so there is no attempt here at generalisation. The planner evolves potential solutions from a function set of unconditional branches and a terminal set of robot actions. Handley's work demonstrates the ability of GP to solve an inverse problem of generating the actions which, when executed, will produce a desired state. There are, however, no functions or terminals which provide any means for the evolved algorithm to reason about the environment at run-time. This is freely admitted by Handley:

> The Genetic Planner does not reason about the world it is planning to act in. Rather, it has a procedural model of the world and it simply *runs* candidate plans to see how well they work.

The work of Handley and others has shown that GP can successfully generate one-off plans. At least with current technology, these examples are infeasible for use in a real-time application as they will take far too long to generate a suitable plan. If GP is to be used in the off-line generation of generalised planning algorithms, previous approaches will need to be enhanced.

One way in which this might be done is to use memory. Usually, when humans solve problems, whether mentally or by computer, they use some store of state or memory in the form of the brain, a piece of paper or computer RAM. In most cases when GP is applied to a problem, there is no explicit use of state or memory which can be manipulated and examined. Andre [1994] discusses the use of memory using a problem involving digging pieces of 'gold' from a grid world. He divides the problem into two stages — a map-making stage and a map-using stage. The most complex problem he solves is for a 4×4 world containing 10 pieces of gold. The map-maker has access to the world, and to the memory. It is able to interrogate the world but cannot manipulate it. The map-user may access the memory and may manipulate the world by moving around and digging. The map-user is not able to 'see' the world directly, only indirectly via the map. A successful solution requires that the map-maker

and user agree on a map representation, and correctly generate and read the map thus collecting all the gold without performing any false 'digs' in squares which are empty.

As was discussed in Section 2.2.1, whilst task planning is important in robotics, the problem which is being focused on in this dissertation is that of path planning. Some of the techniques used in this related area may, however, be useful in GP/GA path planning.

3.11.2 Path Planning

The classical path planning problem has also been tackled by a number of researchers using GA and GP. GA has been used to evolve an order dependent sequence of trajectory translations to allow the end-effector of a 3-dof robot manipulator to follow a path between two points [Davidor, 1989; Davidor, 1991]. Each movement of the robot was specified by a tuple of translations; one for each joint of the arm. The fitness was based on a sum of the deviations of the evolved path from the desired, straight line path between the start and end positions. This approach is easy to apply to a more complex robot manipulator, although there will be an inevitable increase in the computational requirements. It is a one-off planner, in that it plans for a single instance of the path-planning problem and does not seek to generalise to other, unseen instances of the problem.

A number of researchers have worked on an Evolutionary Planner/Navigator (EP/N), a recent description of which can be found in Xiao et al. [1997]. It is argued that standard evolutionary approaches are not able to effectively solve path planning problems, and in response the system is a non-standard GA approach where the problem is divided into an off-line planner and an online navigator. This idea is analogous to the idea of off-line planning, and online obstacle avoidance.

The off-line system evolves a path encoded as a chromosone containing a set of points. Special operators are used to modify individual solutions. Although the more standard mutation and crossover operators are used, these are supplemented with domain specific operators which have been designed with some heuristic knowledge. They use eight different operators, and have achieved better results by adapting the operator probabilities during the run rather than adopting the more usual approach of fixing the operators at the start of the run. They achieve this by measuring the performance of their operators in terms of factors like how effective they are at improving paths, and how costly they are to apply.

Having evolved successful paths off-line, the online navigator uses the pre-evolved paths to manoeuvre in the world. It is able to modify the paths to cope with previously unknown obstacles which are encountered. Whilst this work has many merits, it is still evolving action plans for specific initial and goal positions. It is interesting to note that in their future work, the authors consider, like Andre [1994], Teller [1994] and others, that some concept of memory may be useful in the further advancement of their EP/N algorithm.

An approach used by Zhao and Wang [1998] adopts one of the many variations on GP. They use the Chromosome-Protein Scheme which uses the familiar hierarchical structures, fitness assignment and evolution of GP and GA, but the contents of the nodes of the trees are domain independent, as opposed to domain specific as in GP or GA. The trees contain functions which manipulate domain specific 'amino acids'. In this case the amino acids correspond to direction 'road-signs' which the evolved chromosones placed on a grid representing the workspace to direct a virtual robot around the obstacles to the goal. In [Zhao and Wang, 1998] it is suggested that paths can be successfully evolved, although the time taken to evolve solutions, nor the regularity with which success is achieved are reported.

A recent novel approach adopted by Hocaoğlu and Sanderson [1998] uses an evolutionary approach to evolve a successful path. The representation is an

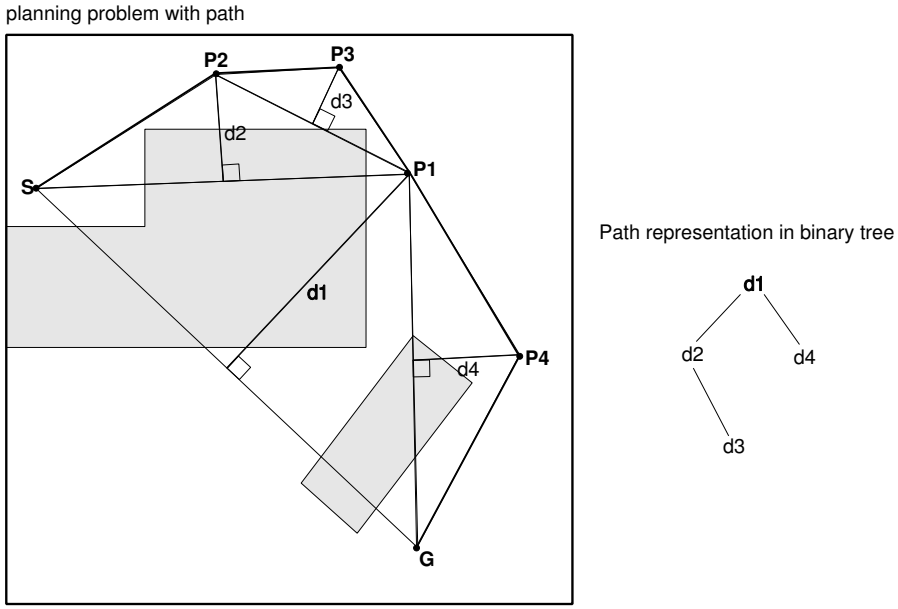


Figure 3.7: Path Representation method used by Hocaoğlu and Sanderson[1998]

ordered set of one-dimensional distances, which are used successively to modify a straight line between the start and goal. The data structure used lies somewhere between that of GA and GP in that it is a hierarchical structure (a binary tree in this case), but all nodes are the same type — there is no differentiation between function and terminal nodes as in GP. The construction of the path is best demonstrated by a diagram as shown in Figure 3.7.

The evolved values d1 to d4 uniquely define points in a path. The root of the tree shown contains d1 which defines point P1 as the end of the perpendicular bisector of the line between S and G — the start and goal. The remainder of the tree is traversed in a pre-order fashion (node first, then left branch, then right branch). Child nodes to the left define extensions to the graph to the left, and child nodes to the right define extensions to the right, as can be seen in the diagram. Once the set of points have been positioned, the path is defined as the result of traversing the tree in-order (left branch, followed by current node, followed by right branch), with the start prepended and the goal appended, resulting in this case in S,P2,P3,P1,P4,G.

This approach simultaneously evolves a set of points and the order in which they should be traversed in a very compact representation. It is, however, limited to planning for a single start, goal combination.

3.11.3 Hybrid Approaches

Sometimes, it is possible to use GA or GP to enhance a pre-existing classical technique. Genetic Algorithms have been used to generate robot motion plans for holonomic and non-holonomic robots. Ahuactzin et al. [1992] encode a sequence of configurations in C-space, or a sequence of move or rotate commands in a bit-string. They also utilise parallelism inherent in GA to speed up their application on a transputer network. A problem with this approach will arise when very long paths are required. The bit-string increases with the length of the path. This work demonstrates the combination of GA and pre-existing motion planning techniques, namely the use of configuration space, to arrive at a successful hybrid technique.

More recently the authors have incorporated their approach into a framework which they have called 'Ariadne's Clew' [Bessière et al., 1993; Mazer et al., 1998]. They use the original work to perform a local 'SEARCH', which is augmented with an 'EXPLORE' algorithm. The Ariadne's Clew algorithm first executes a search to establish whether a simple path exists between the initial and goal positions. If one cannot be found, the EXPLORE algorithm is used to place a 'landmark' somewhere in the workspace. SEARCH is then used to try to find a path between the landmark and goal. If, again, a path cannot be found, the EXPLORE algorithm is used to place another landmark. The landmarks are distributed evenly over the workspace. The SEARCH, EXPLORE cycle is repeated until a path (if one exists) is found.

Genetic Algorithms are used to perform path planning in Artificial Potential Fields (Section 2.9) by Rylatt et al. [1995]. The ability of GA to efficiently search a space is used to find configuration sequences between the known

initial and goal configuration which avoid obstacles in the environment. They conclude that using APFs to characterise the problem space reduces the high computational requirements associated with applying GA to path planning. The emphasis within the research of Rylatt et al. [1995] is on deriving a path from an APF, in contrast with the work presented in Chapter 5 which seeks to improve the quality of the APF itself. Rylatt et al.'s [1995] work could be combined with the work in Chapter 5 in the future construction of a complete planner.

3.12 Summary

The use of Evolutionary Algorithms, and more particularly those of Genetic Algorithms and Genetic Programming have been presented with a view to their application to path planning. Problems associated with the techniques have been highlighted and addressed, in particular the problem of the greedy nature of GP with respect to processor and memory resources. Previous applications of GA and GP to path planning applications have been explained. These applications are all worthy in their own right, but none of them seek to generalise, and in most cases will not be scalable to higher dimensional problems as the computational requirements rapidly increase.

Having identified a domain in path planning, and a technique for tackling this domain, the forthcoming chapters will describe methods in which new approaches to path planning using GA and GP have been implemented.

Despite the drawbacks, it is considered that GA and GP are both, nevertheless, suitable tools to tackle the path planning problem, in light of their previous successful application to planning and other problems. The forthcoming chapters describe new applications of GA and GP to path planning and evaluate the success of these approaches.

4

Evolving a Planner with Genetic Programming

4.1 Introduction

This dissertation has so far identified the need for capable path planning techniques, and has identified shortcomings in traditional approaches. The fields of GP and GA have been suggested as useful methods to apply to the path planning problem. Existing applications of GP which perform planning have been reviewed, but it has been seen that these tend to produce one-off solutions, rather than generalised solutions which can generate paths to problems which they have not necessarily seen before. This chapter explores the idea of implementing a generalised path planning system using GP.

Although the path planning problem used throughout this chapter seems quite simple, the aim of using a computer to automatically evolve a general planner, or planning rule, rather than a single path is rather ambitious. The domain knowledge which a human programmer can add to a manually developed planner is discarded, and the GP process is left to discover the knowledge needed to solve the problem for itself.

In this chapter the path planning problem is explained, and is tackled using a variety of different GP approaches. The first evolves a rule to generate a complete robot plan for a given initial and goal position by outputting instructions which can be directly followed by the robot. The second approach learns

to output a distance transform for a given problem. This involves outputting a 'distance-to-goal' value for each position in workspace, such that the robot can follow these distances in a gradient descent manner to reach the goal. The final approach uses GP to predict the next step to take. For a given position and goal, it outputs the next direction in which the robot should travel in order to move closer to the goal.

4.2 A Path Planning Problem for GP

Having looked at various different planning problems over the last two chapters, it is worth re-stating the problem to be tackled in this dissertation. The aim is to produce a plan to navigate a robot between an initial and goal position in an environment containing obstacles. The ideal situation is to produce a system which takes as inputs, the initial position, goal position and a description of the environment and outputs a plan which can be sent to the robot. A further aim is that the rule should be general in that it should be capable of producing a plan for a similar problem, such as one with different initial conditions or different obstacle positions.

The scenario tackled in this chapter is a problem which involves moving a simple translational robot around a 5×5 grid as shown in Figure 4.1. The grid contains three obstacles at coordinates $(2,4)$, $(3,3)$ and $(4,2)$ and is bounded on all sides by walls which are effectively constructed of obstacles. In this way, the grid is non-toroidal. The robot moves one step at a time north, east, south or west. The problem to be solved is that of moving the robot within the grid from an initial position (xi,yi) to a target position (xg,yg).

Such a small grid makes it easy to solve the problem simply by executing a random walk, however what is required is that the trajectory followed is optimised with respect to the number of moves made by the robot. A move is counted as an attempt by the robot to move in any of the four directions, even if that move would result in a collision with a wall. When a collision

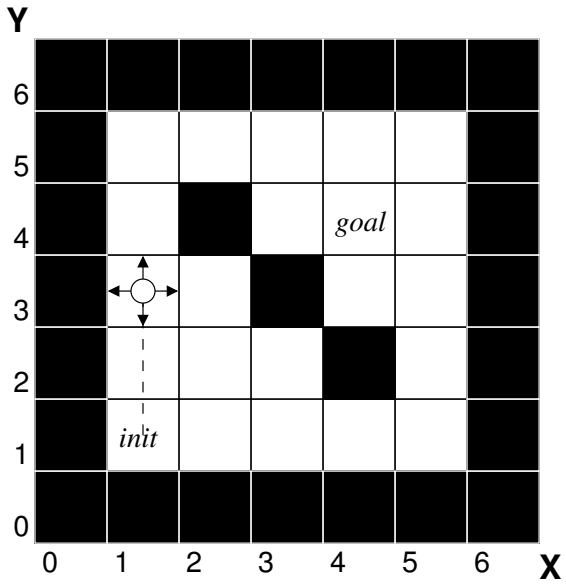


Figure 4.1: 5×5 grid used in simple path planning problem

does occur, a move and a collision are counted, but the robot position remains unchanged. The robot is then allowed to continue. This assumes that in a real application, a collision detection system would be present as a failsafe to prevent invalid plans causing damage.

The aim is therefore to plan the shortest, collision free trajectory from an initial to a goal position. Furthermore, it is required that a solution work not only for a single case of initial and goal position, but for all possible cases. With this small grid, a program can be easily written to generate paths for all 462 cases which exhaustively searches for paths. However, for more complex path planning problems, an exhaustive search may be intractable. Generating a solution to this simple problem without exhaustive search will lay foundations on which to base future research into techniques which may provide the solution to complex path planning problems.

The proposed result of this piece of research is a rule or function which may take some inputs, perform some evaluation and produce corresponding output. This type of problem fits naturally into the remit of GP. GP can be used to evolve solutions which are programs or rules expressed in a language based

on the problem domain. If GA were to be used for this problem, an abstract representation would need to be devised which could be interpreted as a rule which in turn could be used to produce the desired output. It is not the case that GA is not as able as GP at solving problems — indeed they rely on the same underlying mechanisms for their success — rather it is more intuitive to use GP for the human programmer.

In this chapter, which makes a first attempt at addressing the path planning problem, the classical work reviewed in Chapter 2 is somewhat put aside. When applying evolution to solving a problem with a computer, there must be an acknowledgement that the human understanding of the mechanisms which surround the problem at hand are insufficient, or that the application of this understanding is beyond the capabilities of a human programmer. When setting out to solve a problem using GA or GP, care must be taken not to build in too much domain knowledge such that the evolutionary search is in the wrong direction, and prevented from ever finding its own way towards better solutions.

The application of GP to the problems in this chapter was carried out using custom software written in C. In most part runs were executed on a single processor Pentium machine running Linux. On occasions, for large runs, execution was carried out on a small network of Pentium powered Linux machines, or on SUN SPARCstations, using the parallel approach as described previously in Section 3.9. The evaluation of the fitness for the problems was achieved using a simulation and not with feedback from a real robot.

4.3 Evolving a Planner using GP

This section describes an approach whereby GP is used to evolve a rule which inputs the maze, initial position and goal position, and outputs a complete sequence of moves which will successfully navigate the robot along an optimum path between the initial and goal positions.

It has become commonplace to follow the example of Koza [1992] in describing the application of GP to a problem in a *tableau*. The tableau for this problem is shown in Table 4.1, however further explanation of this GP application will be given during the remainder of this section.

Objective:	Evolve a program which generates the actions to move a simple robot on a collision free trajectory between two points on a 5×5 grid.
Terminal set:	*xi*, *yi*, *xg*, *yg*, $\mathcal{R}$
Function set:	`iflt, ifgt, ifeq, ifobs, branch2, plus, minus, north, south, east, west`
Fitness:	Mean distance to goal over all fitness cases — see text for more detail.
Fitness Case:	50 training cases, consisting of a pair of coordinates — an initial and a goal position
Parameters:	population = 200, generations = 600, program size $<= 200$, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal for all cases = 0

Table 4.1: Tableau for Evolving Robot Plans

4.3.1 The GP Approach

If the aim of this work was to produce a one-off solution to a specific initial/goal position problem, the representation scheme could consist only of the robot actions as terminals, which could be combined using some unconditional branching functions. However, this problem requires different outputs from the evolved rule in response to different inputs of initial and goal position. It is, therefore, necessary to include these input arguments in the terminal set, whilst the robot actions are included in the function set. The features which it was decided should be captured in the representation scheme were identified as:

- The initial and goal positions would need to be available in the function set.

- The rule would need a means of 'seeing' the maze with a querying function.
- The rule would require some ability to reason about the information it was receiving from the input to the problem, otherwise there would be a danger that a lookup table would be evolved rather than an 'intelligent', general rule.
- The robot actions needed to be included somewhere in the representation scheme.

Terminal Set

The terminal set is fairly straightforward, containing the four coordinates which define the problem as well as a single random constant $\mathcal{R} \in \{0, 1, 2, 3, 4\}$. The representation scheme is an unusual hybrid. The data type for the programs is integer, however the members of the function set are not what would be classed as normal integer operators.

Function Set

The function set contains four conditional operators: `iflt`, `ifgt`, `ifeq`, `ifobs`. These each have four sub-trees. The first three functions `iflt`, `ifgt` and `ifeq` perform a comparison on the evaluation of the first two sub-trees; less than ($<$), greater than ($>$) or equal to ($=$) respectively; and then execute the third sub-tree if the comparison is true, or the fourth sub-tree if the comparison is false.

The final conditional operator `ifobs` takes the results of the evaluation of the first two sub-trees and treats them as an x and y-coordinate. If there is an obstacle at the position (x,y), then the third sub-tree is executed, otherwise

the fourth sub-tree is executed. The fourth sub-tree is also executed if the co-ordinates fall outside the boundaries of the maze. This operator was included as an attempt to allow the evolved programs to 'see' the maze.

Each of these conditional operators returns the integer value which resulted from the evaluation of the third (TRUE) or fourth (FALSE) subtree.

The next two functions, `plus` and `minus`, return an integer corresponding to sub-tree 1 $+/-$ sub-tree 2.

The next four functions generate the robot actions. Actions are often found in the terminal set as in the artificial ant problem, however for this problem it was decided that movement functions rather than terminals would be used to enable the execution of multiple actions. This approach has been used previously, for example by Andre [1994]. The four action functions therefore each have a single sub-tree. The result of this sub-tree dictates the number of times that the action is executed. Similar functionality could have been achieved by using a two argument function which unconditionally executes one sub-tree n times, where n is determined by evaluating the other sub-tree.

Finally the function `branch2` permits unconditional two-way branching. First it executes sub-tree 1, followed by sub-tree 2. The result of sub-tree 2 is returned for use as an argument by higher level nodes.

4.3.2 Fitness

The difficulty with measuring fitness for this problem, is that there are more than one criteria which need to be met for the problem to be solved — that is, it is a multi-objective function. When travelling in a car, it is nice for the journey to be completed as quickly as possible, but there is also a requirement to conserve fuel as much as possible. It is difficult to meet both these objectives because they can conflict. Not moving at all will conserve fuel, but will not help to complete the journey quickly. Driving very fast will complete

the journey quickly, but because the engine is operating at above its most fuel efficient speed, economy will be compromised. The primary objective for the path planning problem is that the robot should reach the goal position. However, there are also secondary objectives that the robot should minimise the number of steps taken to reach the goal, and should not collide with obstacles during its journey.

The primary aim of making the robot arrive at the goal was achieved by using a distance measure:

Euclidean Distance

The first, and most crude fitness measure used was to compare the distance between the final position of the robot and goal after the execution of the GP generated rule instantiated with specific initial and goal positions. The fitness assigned was $\sqrt{x^2+y^2}$, where x and y are the distances between the start and the goal. Therefore the perfect fitness, when the robot finishes at the goal, is 0.

Distance Transform

The distance transform, or J-function value [Werbos and Pang, 1996] has been used in robot path planning because of its simplicity, and its ability to cope with multiple goals [Zelinsky, 1992]. The distance transform has also found applications in computer vision [Rosin and West, 1995]. Although it is simple to compute it can be computationally expensive for large search spaces. The distance transform in the context of this problem is simply the number of robot steps along the shortest path between the current position and the goal. The distance transform accurately measures the performance of a solution to the problem, because it takes account of the distance which must be travelled to move round an obstacle. Euclidean distance only measures a straight line

passing directly through obstacles if they happen to lie in the line of sight between the robot resting position and the goal. In practice it is much easier to calculate the Euclidean distance, than the distance along the path.

Multi-objective Fitness

For this problem, when using distance alone, good solutions will be encouraged to move towards the goal, but they may not necessarily take a very direct route and they may also collide with obstacles on the way to the goal. To counter this problem the fitness can be made to include a path length term, and a collision term rather than distance alone. However, the inadvertent result is that GP very quickly determines that an efficient way of avoiding collisions and minimising path length is to evolve a rule which causes the robot to remain stationary irrespective of the input. Therefore, by attempting to improve the fitness measure, the result has been quite the opposite.

This non-moving solution occurs because early in the GP run, a relatively good fitness is assigned to those individuals which do not move at all. Because the first generation in the run is generated randomly, it is likely that several non-moving individual programs will exist. The relatively good fitness assigned to these individuals allows them to prevail in favour of other individuals which do move towards the goal and therefore are likely to collide and therefore incur a fitness penalty for path length.

One way of addressing this problem is to apply a multiplier or weight to each fitness term in the fitness vector to indicate the relative importance of each term [Zhao and Wang, 1998]. If the scaling of the secondary fitness terms is relatively low, the population will not be polluted by non-moving individuals early in the run. The scaling must also be sufficiently high such that the fitness terms do have a sufficient effect on the solutions — too low a scaling factor and they will influence the evolutionary process little or not at all.

Another problem to be considered is that each fitness criteria may be capable of returning values from different ranges [Bentley and Wakefield, 1998]. The weighting factors may be doing two jobs: unifying the fitness ranges, and determining relative importance.

A more dynamic approach attempted, was to teach GP to produce good solutions by gradually raising the expected standards over the course of a run. This is achieved by applying a weight to the secondary fitness terms which is a function of the progress of the run, for example $(\frac{g}{g_{max}})^n$. This results in the initial fitness of individuals to be assigned only using distance, but as the run progresses towards final generations, the secondary fitness terms become significant as their weighting increases.

Another dynamic approach which has been tried is to set the scaling factor or weight applied to the secondary fitness terms to be a function of the primary fitness term. This adaptive approach only introduces secondary fitness requirements as performance against the primary fitness improves. If this causes a degradation in the primary fitness term, the importance placed on the secondary terms is played down. This approach is rather like the adaptive learning rates used in neural networks to adjust the rate at which error is propagated back through a network in order to adjust the weights and improve the performance of the network [Noyes, 1997].

4.3.3 Training Cases

For this problem, a single maze was used, that being the 5×5 maze shown in Figure 4.1.

The fitness used was:

$$\frac{1}{cases} \sum_{i=1}^{cases} d_i + w(c_i + m_i) \ ,$$

where d_i is the distance between goal and the robot's resting position for case i and w is the weight applied to the secondary fitness terms c_i and m_i which are the number of collisions and moves incurred by instantiating the evolved rule against fitness case i.

4.3.4 Results

Experiments were carried out primarily to determine whether a good rule could be evolved for this problem. However, the best method of dealing with the multi-objective nature of this problem was also tested.

The tree shown in Figure 4.2[1] represents an example of a rule generated using GP. It was evolved using a population of 200 individuals and arose at generation 978. The fitness of this individual is 0.9675, which is the mean number of grid-squares between the resting position of the robot and the goal.

The full tree shown in Figure 4.2 is uneditted. A manually editted version is shown in Figure 4.3. The unused branches have been removed from this tree, and remaining nodes have been simplified.

The graphs in Figure 4.4 demonstrate the effect on the performance of the best individual from each generation by setting n to 1, 2, 3 and 4. The parameter n determines how quickly the weighting of the moves and collisions parts of the fitness is increased. For the special case where $n = 0$, the weight is held at 1.0, and therefore the secondary factors are given full consideration. In this case, all runs immediately converge to solutions which do not move. For subsequent values of n, the use of the scaling according to run progress seems to postpone this effect. The gradual increase in the consideration of secondary fitness terms eventually causes the sudden emergence of a non-moving individual which then prevails for the remainder of the run.

[1] All trees in this dissertation were plotted using daVinci v2.0.3 [Frölich and Werner, 1996]

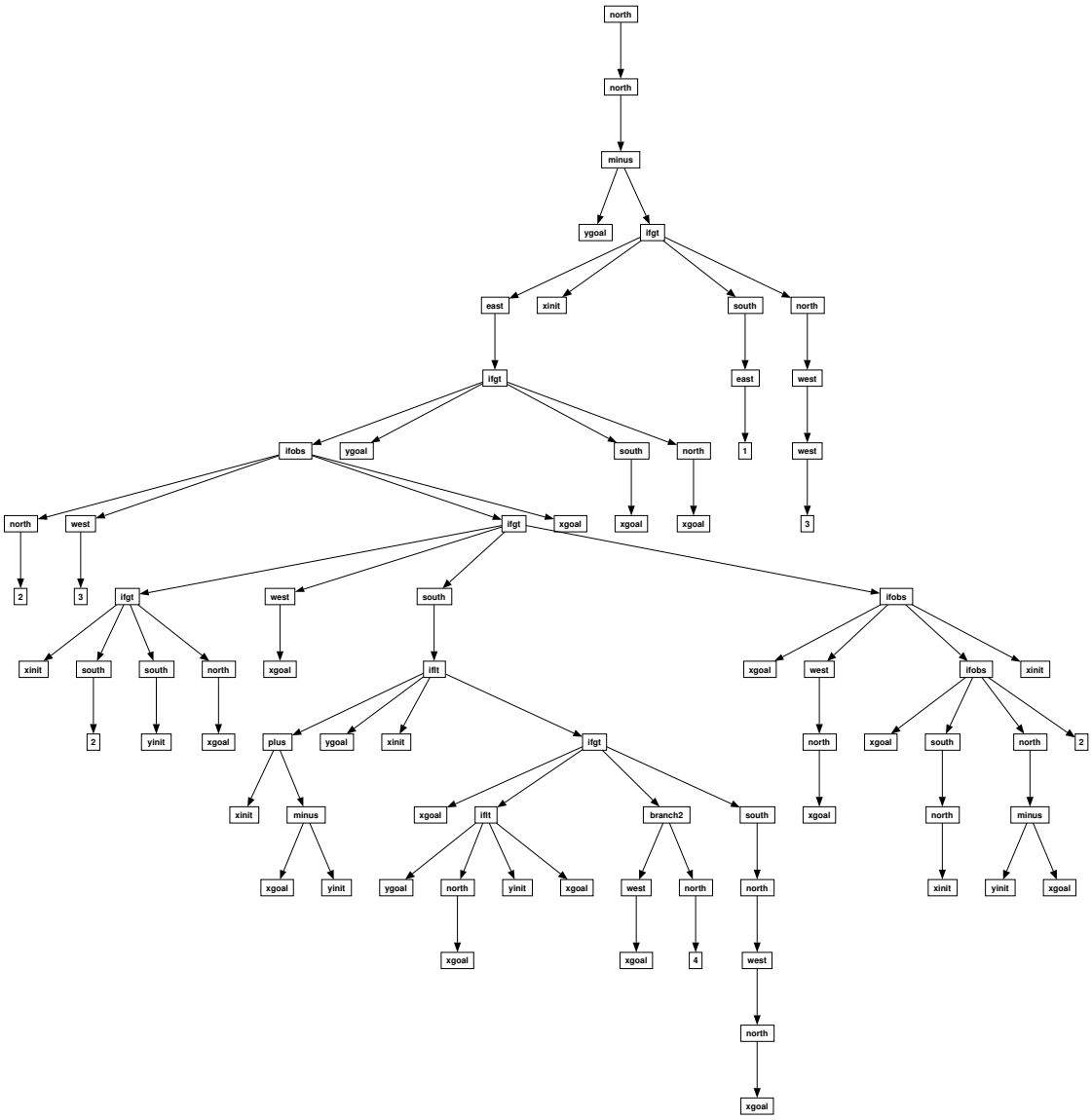


Figure 4.2: Uneditted planner evolved with population size=200

The use of the adaptive weight was more successful, and it was found that, on average, the performance of best individuals incurred less moves and collisions than for the distance only fitness approach, but the mean distance to goal increased. Two graphs are shown in Figure 4.5 which allow comparison between an approach which does not use adaptive weights, and one which does. Each graph shows the mean values from ten differently seeded runs. The runs have only been executed to 50 generations to allow the details to be

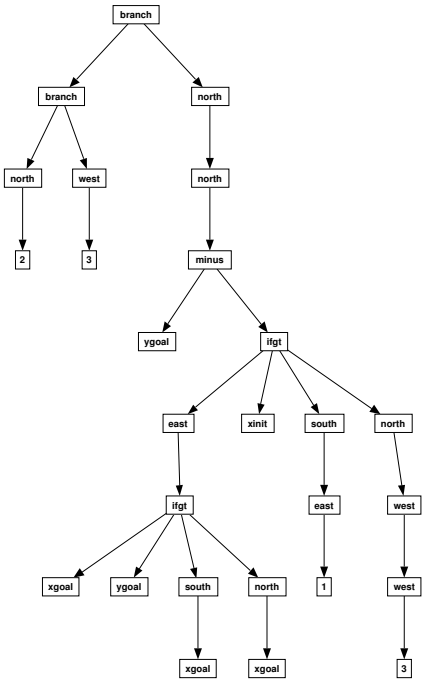
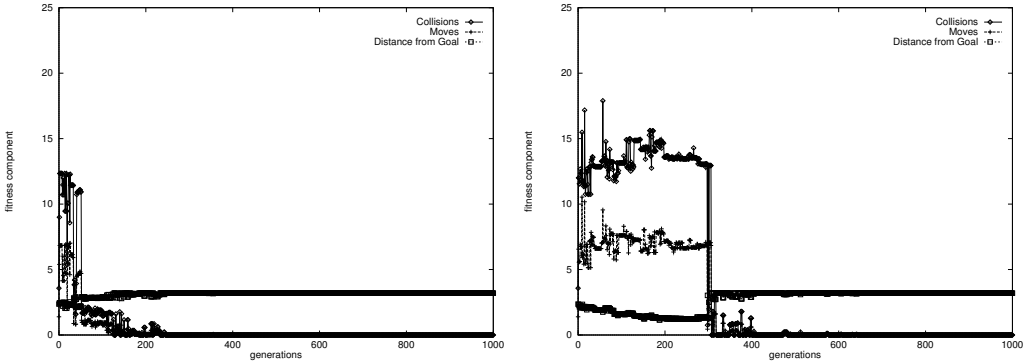


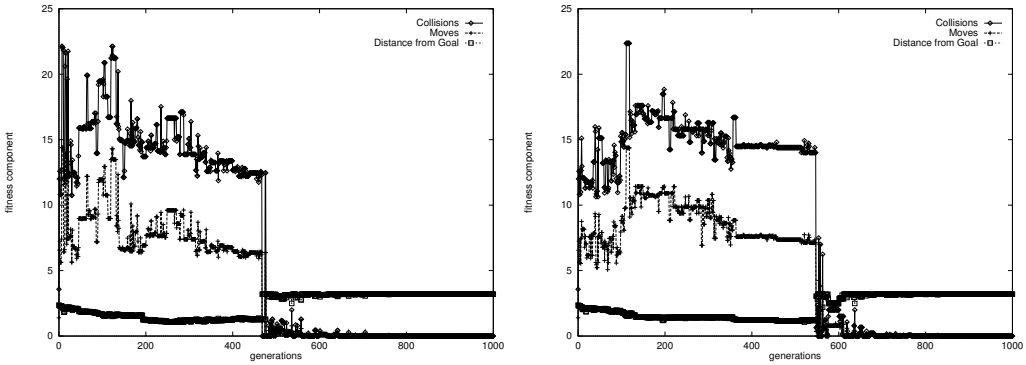
Figure 4.3: Edited planner evolved with population size=200

seen more clearly.

A summary of the best results evolved using various approaches is shown in Table 4.2. The first fitness method uses only the resting distance between the robot and the goal. The second also uses a sum of the distance, the number of collisions and the number of moves made by the robot. No weighting was applied to the secondary criteria. The result is the non-moving individual described above. Results for this experiment are given to allow comparison of the distance to goal achieved. The third method scales the secondary fitness criteria by a function of the generation number. The fourth, and most successful approach is that using adaptive weighting. From this table, it can be seen that the best results come from the use of adaptive weighting of the secondary fitness criteria.

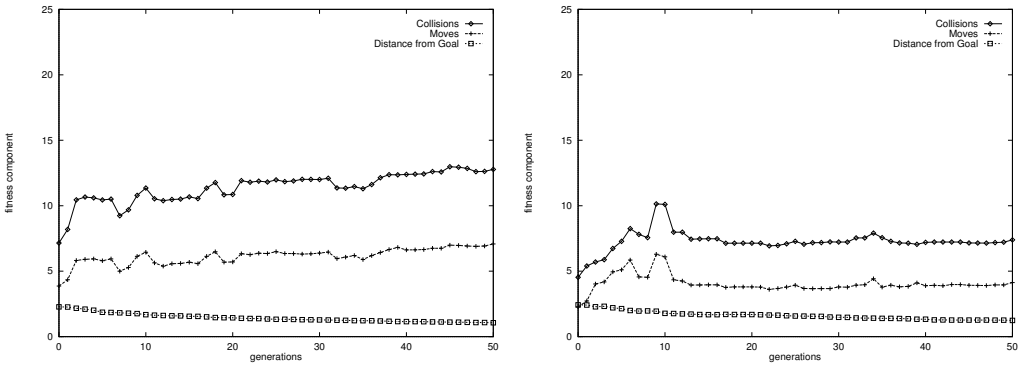


$n = 1$ $n = 2$



$n = 3$ $n = 4$

Figure 4.4: Graphs demonstrating fitness scaling using $(\frac{g}{g_{max}})^n$ for n=1..4



Without Adaptive Weights With Adaptive Weights

Figure 4.5: Demonstration of improvements achieved using adaptive secondary fitness weighting

4.4 Evolving Distance Transform

Given the lack of sufficient success for the previous approach, it was decided that GP would be applied in a different way. This dissertation con-

Fitness Method		SEEN				UNSEEN			
(see key below)	gen.	dist.	collide	moves	% at goal	dist.	collide	moves	% at goal
1	511	0.7811	9.07	14.75	41%	1.2844	8.31	14.53	27%
2	0	2.6769	0.00	0.00	0%	2.9508	0.00	0.00	0%
3	559	2.0700	3.80	7.00	0%	1.8159	3.51	6.73	16%
4	688	1.2443	1.30	4.02	48%	1.1844	1.31	3.59	50%

Key to fitness types:

1. minimise distance to goal only
2. minimise distance + moves + collisions
3. minimise distance + secondary scaled by generation no.
4. minimise distance + adaptive weighted secondary fitness

Table 4.2: Comparison of results of Simple Path Planning Experiments

centrates on evolutionary approaches to path planning, however, it is worth noting that neural networks have also had a part to play in path planning [Harsten, 1990].

Werbos and Pang carried out research on generalised path planning detailed briefly in their published paper [Werbos and Pang, 1996] and in more detail in [Pang and Werbos, 1998]. Their work uses Simultaneous Recurrent Networks (SRNs), and adaptive critics [Barto et al., 1983] which are based on approximate dynamic programming (ADP) — the state-of-the-art in neuro-control. They claim to be able to solve a generalised Maze Navigation problem using SRNs which cannot be solved by more traditional feed-forward or Hebbian networks.

They solve the problem by using a network architecture which places an 11-input, 5-neuron, 5-output net over each grid-square. The inputs indicate whether the square is an obstacle or goal square; they also take input from the four neighbouring cells, and the current cell itself. The weights are not different for each cell, but instead are shared between the nets at all cells. The inputs and outputs are obviously different at different cells. The idea of placing a neural node at each grid cell follows on from work which uses a network of automata to approach a path planning problem [Houillon and

Caron, 1993].

The problem to be solved does not appear complex, but it is correctly argued by Werbos and Pang [1996] that it is valid to attempt such a problem, because if it cannot be solved then there is little point trying to solve more complex problems. He draws an analogy between this 5×5 maze and the XOR problem, which for some time posed a problem in the field of neural networks because it is a linearly inseparable problem which could not be solved by the simple perceptron [Minsky and Papert, 1969]. It is therefore justified to use what appears to be a simple problem as the focus of the experiments.

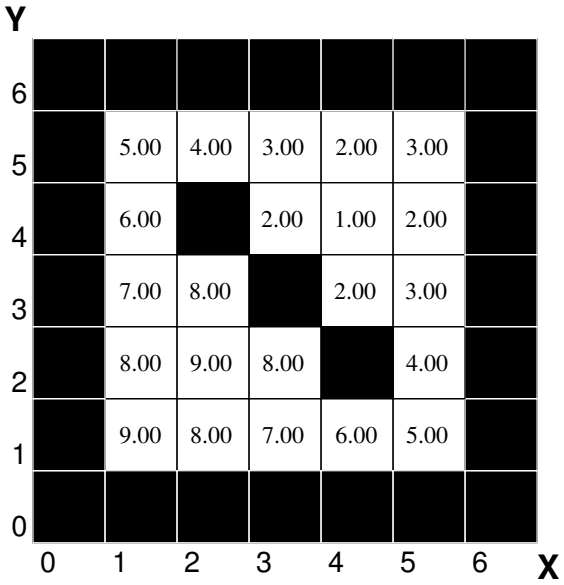


Figure 4.6: 5×5 grid overlaid with a J function landscape

This section presents the use of GP to learn to approximate a function which maps a tuple (x, y, xg, yg) to a distance representing the shortest number of steps between the current position (x, y) and the goal (xg, yg); that is, the distance transform discussed earlier. This distance is referred to in this section as J as per the Werbos and Pang work. The problem is again applied to a 5×5 grid. Figure 4.6 shows such a grid overlaid with values representing a J-landscape for $xg = 4$, $yg = 4$. There are a total of 22 such grids, of which this is one example. The intention is to generalise by training a rule using

only a subset of the total possible examples, which can later cope with unseen inputs.

It has been noted previously that the distance transform is more difficult to determine than the more straightforward Euclidean distance. If, however, a successful rule can be trained using relatively few training examples, the time saved may justify the approach. There are 462 possible initial and goal position combinations for a 5×5 maze alone. If a successful rule can be evolved with 10 or 20% of these combinations, and the time taken to evolve the rule is less than the time taken to compute using a brute force approach the remaining 80–90%, then it is useful to use GP.

4.4.1 The GP Approach

The evolution of a distance transform is, in essence, a function approximation problem. Unlike the previous approach, it is easier to find a function and terminal set which fits well with the domain. In fact the domain has been changed from a robot action oriented approach to a functional approach. A layer has, therefore, been added to the problem, as solving the problem no longer involves directly executing the evolved rule. Instead, a simple algorithm follows the direction of steepest descent of the function evolved. Once again a tableau is given Table 4.3. This approach has another appealing advantage, given the problems encountered previously, in that multi-objective fitness is not required.

The terminal set provides a set of inputs consisting of two pairs of coordinates for the current and goal position. It also contains two other symbols, OBS and GOAL. These are effectively boolean variables taking the value 1.0 if the current position, defined by (x, y), coincides with an obstacle or the goal respectively. For any other grid position, these symbols are set to 0.0. The terminal set also contains a random real variable which may take a value in the range 0.0–5.0.

Objective:	Evolve an algorithm to output the distance transform for a given current position and goal for a translational robot in a 5×5 maze.
Terminal set:	*xpos*, *ypos*, *xg*, *yg*, *OBS*, *GOAL*, $\mathcal{R}$
Function set:	`+, -, /, *`
Fitness:	Mean Square error over fitness cases.
Fitness Cases:	Between 1 and 23 grids of distance to goal values — see text for more detail.
Parameters:	population = 200, generations = 1000, program size <= 150, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Mean Error = 0

Table 4.3: Tableau for Approximation of 'distance transform' problem

4.4.2 Fitness

The fitness measure used was a mean square error, that is:

$$\frac{1}{n}\sum_{i=1}^{n}(J_a - J_d)^2 \,,$$

where J_d is the desired value from the training data, and J_a is the actual value generated by the evolved algorithm. The number of data values n varied according to the number of training grids used — $n = 22$ for a single grid to $n = 506$ when all the available data is used.

The available training data for this problem comprised a set of 23 grids of the type shown in Figure 4.6; one for each of the 23 possible goal positions. Various runs were carried out varying the amount of training data.

4.4.3 Results

A series of runs were executed using training data consisting of 1, 5 and 8 training grids. Training for a single goal as per the Werbos and Pang experiments, a per square error of 0.66 was achieved. This number does not really mean very much, so it is better to see the resulting grid in Figure 4.7 and compare it with the correct grid in Figure 4.6. Comparing these results with

Pang and Werbos's [1998] report, the error is significantly better than their use of a Simultaneous Recurrent Network (SRN) trained using truncation, for which the mean error per cell was 2.30. The GP approach, however, did not perform better than their use of an SRN trained using Back Propagation Through Time (BTT) for which the mean error per cell was only 0.18.

When trained against 5 grids of data, the performance worsened, yielding a mean error of 1.53 per grid square. When trained against 8 grids, the performance decreased further to 2.68. This progressive worsening of the results conflicts with the intuitive view of supervised learning, the reason for this is discussed in chapter 6. As the number of training grids increased, a tendency towards a homogeneity in the J values output can be observed. An example of this for the 5 grid experiments can be seen in Figure 4.7, as can the extreme case for the 8 grid experiments.

Avoiding Obstacle Evaluation

If it were not for the obstacles in the environment, this path planning problem would be made very much easier. Obstacles represent discontinuities in the 'distance to goal' landscape, and are therefore difficult to deal with. The approach used was to assign a high 'distance to goal' value for an obstacle. The experiments here have also used this approach by attaching a value of 10 to obstacles — a value which is known to be higher than any other valid distance to goal.

Using this technique meant that sharp peaks appear in the landscape generated by the evolved GP function. When the target position is adjacent to an obstacle, a high gradient must exist between two adjacent points in the function, and when the target is further away from an obstacle, the gradients are gentler. The evolved algorithm must capture this requirement to adjust the gradient rapidly to small changes in domain variables.

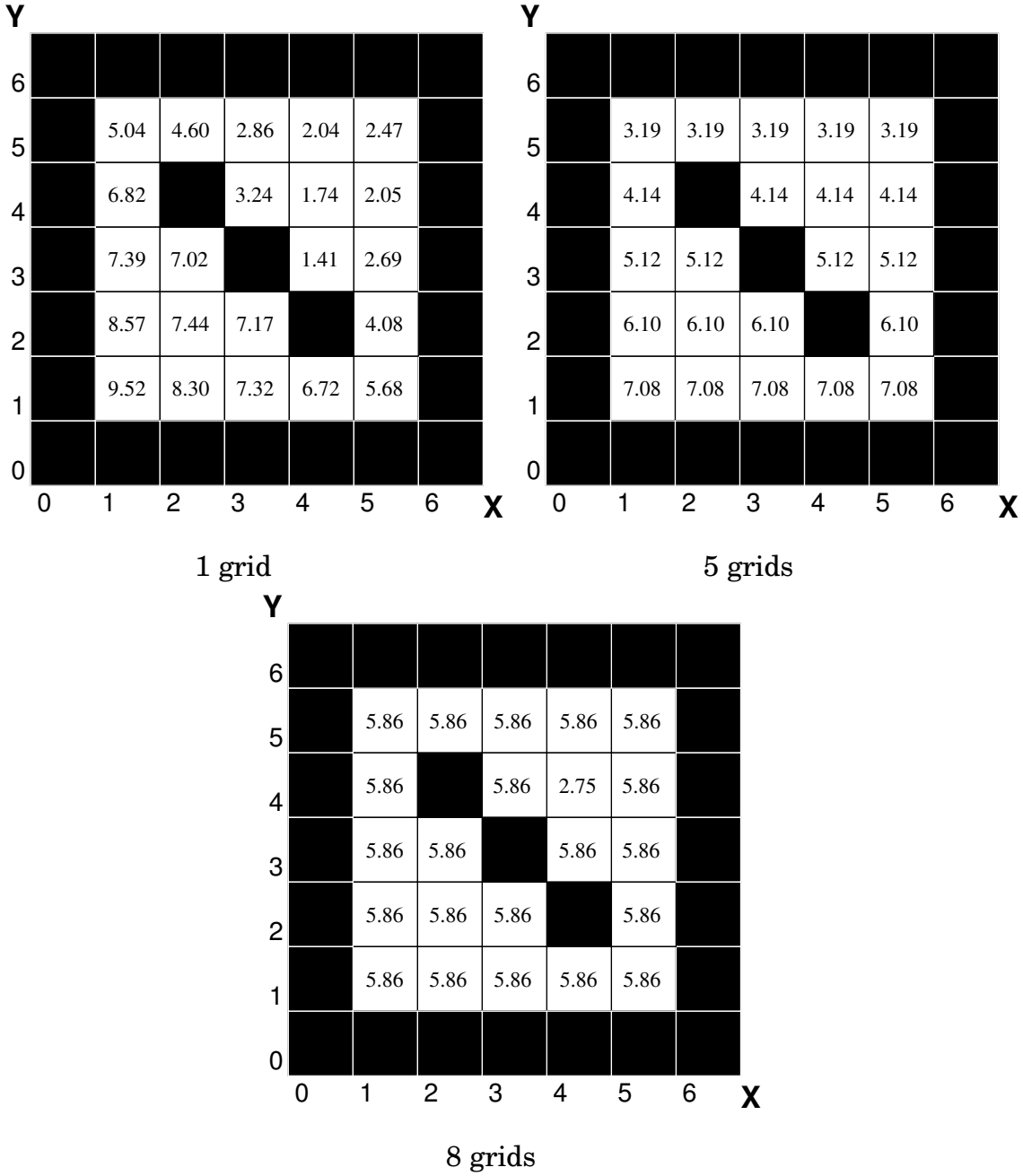


Figure 4.7: GP generated J function evolved with 1, 5 and 8 grids of training data

	With Obstacles		Without Obstacles	
Seed	Fitness	at gen.	Fitness	at gen.
1	10.928	68	9.755	494
2	9.574	497	9.325	377
3	9.857	476	10.042	459
4	10.095	496	10.089	81
5	10.971	25	8.700	398
6	9.171	347	8.874	274
7	10.084	423	8.336	493
8	8.262	456	10.565	2
9	10.090	265	10.095	499
10	10.083	175	10.095	451
11	9.320	311	10.655	8
12	10.098	99	8.920	174
Mean	9.878		9.621	

Table 4.4: Results of Obstacle Evaluation comparison runs

A simple experiment was carried out to look at the difference in performance of the system when evaluating GP individuals at obstacle coordinates, and when excluding evaluation at these discontinuities. Two sets of runs were executed with the intention of collecting the fitness of the best individual from each run and to try and draw conclusions as to which method was best.

The results of these runs are shown in Table 4.4. Twelve, differently seeded runs were started for each scenario. The fitness of the best individual from each run, and the generation number at which that individual first appeared is given. The fitness reported for each run is the mean sum of the squared errors over a 5×5 grid. So, for example 10.928 represents a mean square error of $10.925/25 = 0.437$ per grid square. It can be shown using, the t-test [Rees, 1989], with 95% confidence, that the mean fitness from the two methods are equal, therefore there does not seem to be any advantage in using one method over the other.

Whilst it was hoped that one method would result in more fit individuals than the other, this could not be demonstrated. However the similarity in the results does confirm that it is not necessary to evaluate at obstacles, which

means a time saving can be gained without degrading the performance of the algorithm.

4.5 Evolving a Next Step Predictor

The evolution of a distance transform function yielded more positive results than the direct evolution of a robot program, however there was still scope for improvement. The final alternative presented in this section again uses a function approximation approach, similarly to the distance transform approach described previously. The function which is sought to be approximated in this approach is one which maps an input tuple $(xpos, ypos, xg, yg)$ as per the previous problem, and outputs a number in the interval $[0..3]$ corresponding respectively to the robot moving north, west, south or east — whichever direction results in the robot moving along an optimal path from the current position towards the goal.

The idea is demonstrated in Figure 4.8 where the top left grid shows the distance transform for a problem where the goal is at $(4,4)$. The top right grid shows the corresponding direction in which the robot should move in order to follow the optimum path to the goal. Finally, in the bottom right of the figure, the numerical values corresponding to the direction are shown. It should be noted that in some cases there are more than one direction which are equally good. When this situation occurs, the lower number is chosen — this is an arbitrary choice.

4.5.1 The GP Approach

The setup for this problem is very similar to the previous distance transform approach. The main difference is the difference in the training data. The tableau for this problem is given in Table 4.5.

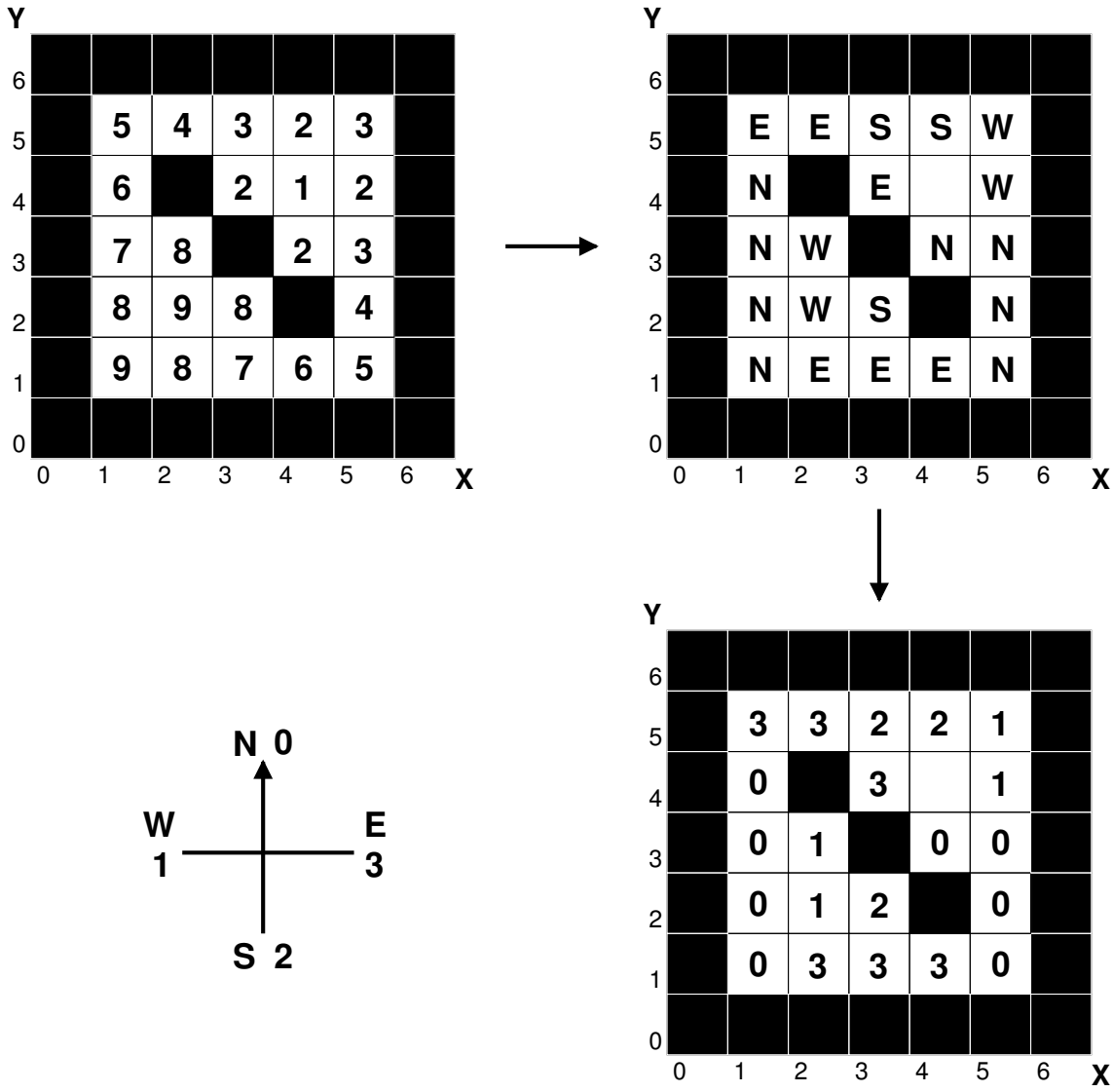


Figure 4.8: Diagrammatic explanation of the next step function

Objective:	Evolve an algorithm to output a number indicating the direction of the next step which should be taken by a translational robot to travel along the optimal path between two points in a 5×5 maze.
Terminal set:	*xpos*, *ypos*, *xg*, *yg*, *OBS*, *GOAL*, $\mathcal{R}$
Function set:	`+, -, /, *, cos, sin, pow, exp`
Fitness:	Function of the optimum number of moves for the given fitness case (See Section 4.5.2)
Fitness Cases:	174 initial/goal position pairs, i.e. journeys
Parameters:	population = 200, generations = 60, program size <= 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Mean Fitness = 0

Table 4.5: Tableau for Next Step Prediction problem

4.5.2 Fitness

A new fitness function was used to take advantage of the full knowledge now available to the evolutionary process. The fitness is the sum of the differences between the distance values along the optimum desired path and the actual path taken by the robot.

$$Fitness = \sum_{t=1}^{t=T} D(t) \, ,$$

where T is the total number of moves and D is a distance function.

The total number of moves is defined as,

$$T = \begin{cases} m_a & \text{when } m_a > m_d \\ m_d & \text{when } m_a \leq m_d \end{cases} \, ,$$

where m_a and m_d are the actual and desired number of moves for the current fitness case.

The distance function is defined as,

$$D(t) = \begin{cases} J(x_a(t), y_a(t)) - J(x_d(t), y_d(t)) & \text{when } t \leq m_a \\ J(x_a(m_a), y_a(m_a)) - J(x_d(t), y_d(t)) & \text{when } t > m_a \end{cases}$$

where $x_a(t)$, $y_a(t)$, $x_d(t)$, $y_d(t)$ are the coordinates of the position on the actual and desired paths respectively at time t, and $J(x,y)$ is the number of moves from position (x,y) to the goal.

4.5.3 Results

The results for this final GP approach are shown in Figure 4.9. The results are presented by showing three examples selected from the best run made. Over all cases, a path to the goal could be found for 66% of initial positions in the training set, and 45% of initial positions in the validation set. The three grids in Figure 4.9 show results for three different goal positions — $(1,1)$, $(5,4)$ and $(3,4)$. The grey grid squares highlight positions on that grid from where a successful path to the goal can be found. Those grid positions marked with an asterisk are ones which appeared in the training set.

Particularly given that the size of environment used for these experiments is only 5×5, it was hoped that this approach would perform much better. It very much seemed that the fitness and representation scheme for this approach did not enable the obstacles to be seen sufficiently. A neural approach taken by Dracopoulos [1998] gave rise to considerably better results. This is perhaps due to the environment being used as an intrinsic part of the neural network architecture, in a similar way to Werbos and Pang's [1996] approach described earlier in this chapter.

4.6 Summary

This chapter has presented a number of approaches in which GP has been applied to a simple path planning. On this particular problem, it seems evident that the neural approaches of both Werbos and Pang [1996] on the distance transform problem, and Dracopoulos [1998] on the next step prediction problem perform somewhat better than the evolutionary approaches used here.

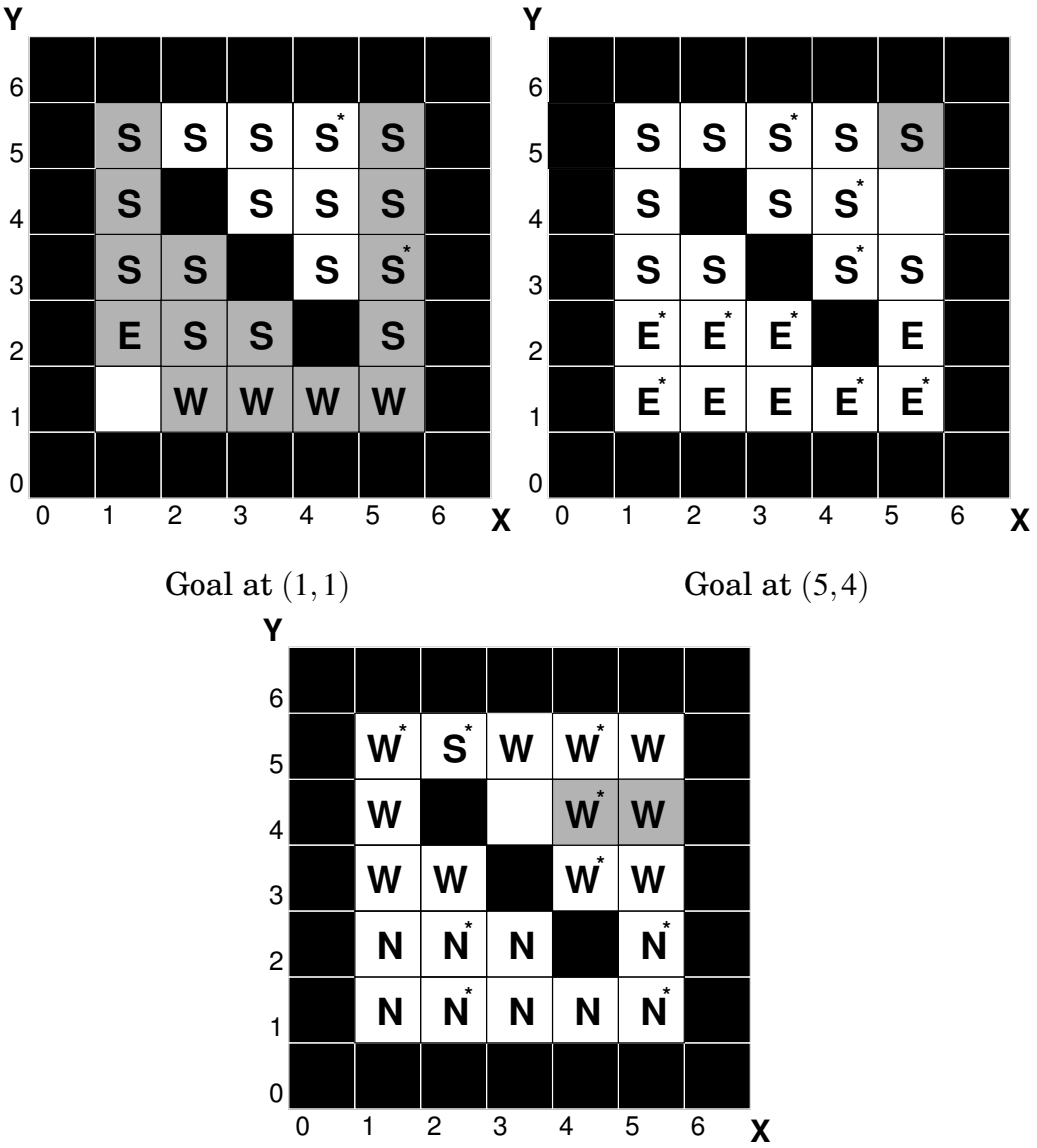


Figure 4.9: Results of the next step prediction path planner

Even these neural approaches are not without their fault. In all cases, a simple 5×5 grid has been used for experiments, and it is easy to imagine the increase in memory and computational requirements which will be experienced when scaling the problem up to something closer to a real problem. In personal communication Werbos [1998] states that performance degraded when attempts were made to generalise using his approach. The inevitable training time versus performance dilemma came into play. Although in this case, neural approaches proved to be better than evolutionary ones, they are both very

susceptible to the effects of making the problem more difficult. The evolutionary approach is not implicitly bound to the dimensionality of the problem as is the neural approach.

The reason for applying neural or evolutionary approaches to the path planning problem is that humans lack a sufficient understanding of how to solve the problem to be able to design an algorithm themselves. It is, therefore, difficult to know exactly why the approaches adopted did not perform as well as expected. The best explanation that can be given for the evolutionary approaches is that they cannot adequately 'see' the environment. With the neural approach, the network was effectively overlaid on the grid, enabling it to 'see' the whole of the environment. Functions were included in the GP approach to query the environment, but this proved to be insufficient. It may be possible to *input* the environment in the terminal set of the representation scheme, but this will dramatically increase the size of the terminal set, resulting in an explosion in the size of the search space for the problem.

Although none of the attempted approaches demonstrated the performance hoped for, the most promising results came from the function approximation approaches. In the meantime, the failings of these approaches will be acknowledged, and a different direction is taken. In the next chapter, an alternative approach will be adopted, whereby a successful classical approach is augmented by evolutionary approaches.

5

Genetic Algorithm Optimisation of Artificial Potential Fields

5.1 Introduction

At the end of the previous chapter, it was acknowledged that the application of GP alone, although providing useful results, unfortunately did not offer a suitable foundation on which to build a full robotics planning system. This was because suitable means could not be found to enable the evolved algorithm to 'see' the environment for which it was planning.

The best results from the Chapter 4 were from the function approximation approaches. One of these evolved a function to output a distance transform approximation. This distance transform is a sort of artificial potential field (APF) laid over the workspace which a planner can follow to reach the goal from any position. This method of using an artificial potential field has been used previously for robot path planning and obstacle avoidance, and was described in Section 2.9.

Having discarded all classical approaches in the previous chapter, this chapter seeks to combine an evolutionary technique, GA, with a classical path planning technique, APF path planning. The GA is used to try to improve

shortcomings in the basic APF technique. The techniques adopted are described, presenting problems which arose and the manner in which they were addressed. In Chapter 4, a single maze was used throughout, as different GP approaches were adopted to try and achieve a successful planner. In this chapter, success is much more forthcoming, and therefore results are given showing how the proposed planner performs in different environments of varying difficulty.

5.2 Artificial Potential Fields

Artificial potential fields [Khatib, 1986; Khatib, 1980] are constructed in a virtual robot environment by assuming a positive charge is placed on the robot and the obstacles in the environment, and a negative charge is positioned at the goal. This has the effect of attracting the robot towards the goal, while repelling it from the obstacles.

The potential field acts on a single point, and whilst many robots do not conform to this specification, it is, however, possible to reduce more complicated robots to a single point problem by using the C-Space approach (see Section 2.5.1). Alternatively, a robot can be reduced to a number of control points, situated, for example at the joint positions of a robot manipulator. In this case a potential field is constructed for each control point, and the effects of all the potential fields on the control points are considered simultaneously in order to move the robot.

The main problem with the APF technique is that the theory that the robot will be pulled towards the goal without collision is not always borne out in practice. Although the robot does not collide with obstacles in the workspace, it can become 'stuck' part way through its journey because the attractive and repulsive forces at a particular configuration cancel each other and cause the robot to halt, or to oscillate around a few adjacent configurations. This situa-

tion occurs at a **well** in the field which does not coincide with the goal — that is, it is a local minimum, rather than a global minimum.

Despite this drawback, the technique is very attractive due to its simplicity and the speed with which the necessary computation can be performed. In theory there are techniques, such as exhaustive search, which are guaranteed to find a global solution to any path planning problem if one exists, but are unable to produce timely results.

This chapter describes an approach which adopts the positive points of the APF approach and which uses GA to reduce the drawbacks.

5.3 Potential Fields for Global Planning

Although the original use of potential fields was for obstacle avoidance, they have also been used for motion planning [Latombe, 1991] as they are easy and fast to compute. Although potential fields offer an appealing means to perform path planning, they suffer from the problem of spurious local minima, because they are essentially local methods. This means that during its route towards the goal, a robot may halt prematurely at a point in the workspace where the potential is lower than in the local vicinity, but is not the lowest in the workspace. For global planning there should only be a single minimum point at the goal — the global minimum.

5.3.1 Numerical and Analytical Approaches

Methods of constructing artificial potential fields can be broadly broken down into numerical and analytical approaches. An analytical approach involves using a function which defines the APF, and to yield any specific value in the potential field, the function parameters are instantiated to represent a given configuration. A numerical approach, on the other hand, involves generating

a lookup table of potential values by processing the configuration space, or workspace of the problem domain.

In Section 4.4, GP was used to approximate a distance transform by evolving a function which, when supplied with inputs would generate a corresponding distance-to-goal value. The function being evolved is very similar to an analytical potential field function. During the training of this function, sample data was provided which the GP process sought to approximate. This training data was generated numerically, by exhaustively processing the path planning environment. A simple algorithm was used which labelled the goal as '1', and used a wavefront expansion process to recursively label all the neighbours of the current cell, incrementing the label at each square by one as it moved further from the goal.

The result of such a numerical approach is a potential field from which a global navigation function can be very easily constructed. The steepest negative gradient is followed from starting position to the goal. If no path exists from the initial position to the goal, this can be immediately recognised because the wavefront expansion will not have been able to assign a potential value to an inaccessible square, in the same way as a fill algorithm in a graphics package will not fill inaccessible areas of the picture.

The simplicity and effectiveness of the numerical approach makes it an excellent choice for some path planning problems. However, as the environment grows, the effort required to numerically generate the field will grow. For the small 5×5 maze used in Chapter 4, the construction of this field took only a few seconds, even when repeated 22 times for each possible goal. As the workspace grows to three or more dimensions, the approach loses its appeal as the computational requirements increase explosively [Latombe, 1991, Chapter 7].

The analytical approach is appealing in that a single function can be used to generate only the values in the potential field which are required to make

the path, and not the values for the whole grid, as required for wavefront expansion. Typically values will be required for all configurations on, and adjacent to, the optimum path. The problem with the analytical approach is that which has been repeatedly mentioned — the occurrence of local minima in the potential field. This chapter describes how GA can be used to minimise the drawbacks of APF for path planning.

5.3.2 Global Navigation Function

When performing path planning, what is aspired for is a global navigation function — that is a function which maps all the free space in the workspace to a real potential value, and has a single minimum located at the goal [Latombe, 1991, Chapter 7].

It would be desirable to have an analytical function which has only a single minimum which occurs at the goal. Unfortunately, it has been shown that, in general, an analytical function cannot be made which generates a potential field with only a single minimum [Koditschek, 1987]. He has demonstrated that, with certain specific worlds containing elliptical or spherical objects, a potential function must contain at least as many saddle points as there are obstacles.

Despite the spurious minima inevitably present in artificial potential fields, if the effect of these drawbacks can be reduced sufficiently, then the speed, and ease with which the APF approach can be applied still allows it to offer an attractive method for use in global planning.

5.4 A new Method for improving Artificial Potential Fields

A stock function for creating an APF is that originated by [Khatib, 1986]:

$$U_{art} = U_{x_d}(x) + U_O(x) \ ,$$

where the value of the potential field at a point x is a combination of an attractive well centred at goal x_d, and a sum of repulsive potentials around obstacles O; $U_{x_d}(x)$ and $U_O(x)$ respectively.

Expanding this further:

$$U_{x_d}(x) = \frac{1}{2}\xi(x - x_d)^2 \,,$$

and:

$$U_o(x) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho} - \frac{1}{{\rho_0}^2}) & if\ \rho \leq \rho_0 \\ 0 & if\ \rho > \rho_0 \end{cases}$$

These definitions have introduced a number of variables which are control parameters which modify the behaviour of the equations:

- ξ Gain of the attractive well, which influences the force with which the robot is pulled towards the goal.
- ρ The minimum distance between the robot and the edges of the obstacle.
- ρ_0 The limit distance around an obstacle, outside which the repulsive effect has no influence on the robot.
- η Gain of the repulsive FIRAS (force inducing an artificial repulsion from the surface — as previously described in Section 2.9.1) function, which influences the force with which a robot is repelled from a given obstacle.

Different settings of these variables result in different potential fields, even for the same environment. Therefore, the settings of the variables influence the existence and number of local minima. For example, if the limit distance ρ_0 of an obstacle is excessively large, and so too is the repulsive gain of that obstacle, a robot may well not be able to circumnavigate the large peak in the potential field to reach the goal. This example is an extreme case, but

demonstrates that the parameters have an influence on the presence of unwanted minima, and by selecting appropriate values for the parameters, these minima can be reduced. In the literature, there is a notable absence of suggestions of methods for assigning values to these parameters, which is surprising given their direct influence on the nature of the APF. Latombe [1991] suggests some rules for setting the parameters in a world containing spherical obstacles, but in general, few papers make such suggestions. There does not seem to be a common algorithmic way in which the gain and distance of influence constants are set. It has been confirmed by Khatib [1998] that no general method existed for setting parameters. Instead, it seems, an informal, heuristic approach is followed whereby common sense rules are adopted, for example, the goal should not lie within the distance of influence of a repulsive field, and field gains should not be unnecessarily high so as to result in very steep gradients. Experiments described in Section 5.11 suggest that the manual setting of these parameters may not be as easy as it would initially appear.

What seems to be lacking is a general method for optimising the parameters of the APF generating function in its application to the path planning problem. The previous attempt at using evolutionary algorithms, in the form of GP, to approximate an appropriate APF yielded only partially successful results. The remainder of this chapter describes how the more traditional method of generating a potential field can be augmented by evolutionary algorithms, this time in the form of GA, by optimising the controlling parameters of the parabolic well and FIRAS functions.

5.5 A Case for Genetic Algorithms

As has been mentioned before there is little difference between GA and GP as far as their underlying principles are concerned. They are both techniques driven by natural evolutionary pressures, and for this thesis, the difference is

considered to be in the data structure evolved and little more. For this path planning approach, GA has been chosen rather than GP. The main reason for this is that the problem more naturally fits the GA. This approach involves optimally setting a number of parameters. If a rule, or function were required, then GP would be the natural choice, however for this problem, the underlying functions already exist in the form of the attractive well, and the FIRAS functions. This approach seeks to make these functions work in an optimal way, by using GA to set the parameters.

Having made the choice of GA from a data structure point of view, the approach inherits certain other advantages. The data structure is a string, rather than a tree, and as such is a much simpler entity to work with. The program code to manipulate the string is simpler, and the evaluation of fitness is also simpler because the simulation involves instantiating the evolved variables rather than constructing and running a different program or function for every individual.

For this problem there is a much lower computational and memory cost than was found with the previous evolutionary approaches which were described in Chapter 4. The basic GA approach is to use a fixed length string to evolve the set of variable parameters which can be set for the attractive well and FIRAS functions described above. The aim is to achieve a resulting potential field which offers an almost global navigation function which can be used for effective path planning.

5.6 Parameters

There are two sets of parameters which can be set; those associated with the attractive well, and those associated with the FIRAS function. For the attractive potential, the only parameter which can be adjusted is the gain ξ.

For the FIRAS function, there are two parameters: ρ_0 and η. One approach would be to evolve only these three values, but it was decided that it would

be more useful to evolve a ρ_0, η pair for each obstacle in the work space. This gives the GA greater flexibility when setting the values which will influence the potential field for the environment. Particularly as the number of obstacles in the environment increases, the use of a single, global ρ_0 and η offers a very broad brush approach, unable to cater for subtleties in the environment.

5.6.1 Solution Representation

The representation of the parameters in a GA string is very straightforward. Each value is represented as an n-bit integer, which is converted to a floating point representation and scaled to lie between two floating point values. Values for the gain parameters lie between 0.0 and 50.0 for obstacles and 0.0 and 100.0 for the goal, and values for the distance of influence lie between 0.0 and 1.0, where 1.0 represents the full width or height of the workspace in normalised space.

5.7 Simulation

A simulation was used to train and evaluate solutions. The simulation used was a custom built system operating on a 2D workspace. An important, early design decision was how to represent the workspace.

Some path planning examples, such as that of Li et al. [1990] use a bitmap representation in which obstacles are represented as filled pixels in an array of appropriate dimension and size. This approach is easy to implement, requiring the declaration of a portion of memory, and setting of the bits accordingly. Another advantage is that collision detection is particularly easy, requiring a simple comparison for a given position in the workspace to determine whether that position is in free space or obstacle space.

The overriding problem with a bitmap representation, is that the memory required grows very quickly as the size, resolution or dimensionality of the

workspace increase. The experiments in this dissertation concentrate on the 2-dimensional case, but it is important from the outset to consider how techniques developed here might scale to 3D problems with higher degree of freedom robots. It was, therefore, decided that a polygonal representation would be adopted.

The advantage of a polygon approach is that the memory required for representation is very compact. Each obstacle is represented by a polygon. Polygon primitives can be defined for different shaped obstacles, and positioned on the workspace via a simple graphics pipeline as found in a more complex form in graphical programming languages such as PHIGS [Hopgood and Duce, 1991]. This applies appropriate transformations to the primitive polygon to rotate, translate and scale the polygon. The obstacle polygons are selected from a small library of primitives. Although in a real-world scenario this might not be sufficient, as unusual obstacles could be difficult to approximate using only the library polygons, it is quite adequate for this research. It makes the definition of a problem very compact. Examples of the problem definition files, and their corresponding environments are shown in Appendix B.

5.7.1 Polygon Collision Detection

When the workspace is represented as a bitmap, collision detection is very simple. The point under test is queried, and if it is set as an obstacle cell, then a collision is found. When obstacles are represented as polygons, the process is slightly more complicated. In this case some vector mathematics can be used. It is assumed that the point under test is P, and that the polygon is defined by a number of points v_n where $n = [1..N]$ (N=number of sides of polygon). These points are to be taken in the anti-clockwise direction.

The cross product of two vectors $\overline{Pv_n}$ and $\overline{v_n v_{n+1}}$ is a line perpendicular to the plane defined by the two vectors. If $\overline{Pv_n} = (a_1, a_2, a_3)$ and $\overline{v_n v_{n+1}} = (b_1, b_2, b3)$, then the cross product is defined as:

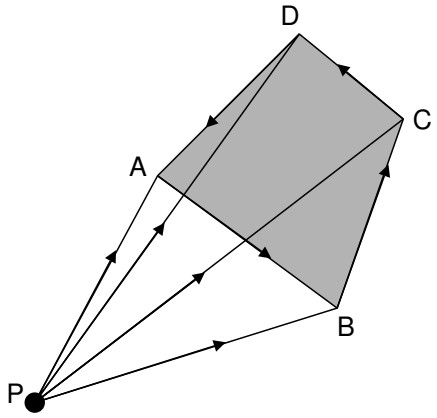


Figure 5.1: Illustration of Polygon Collision Detection

$$\overline{Pv_n} \times \overline{v_n v_{n+1}} = \mathbf{i}(a_2 b_3) + \mathbf{j}(a_3 b_1) + \mathbf{k}(a_1 b_2) - \mathbf{i}(a_3 b_2) - \mathbf{j}(a_1 b_3) - \mathbf{k}(a_2 b_1)$$

where **i**, **j** and **k** are the unit vectors along the x, y and z axes respectively.

If the z-component (**k**) of the cross product is positive for **all** edges n, then the point P must be inside the obstacle, and correspondingly a collision has occurred. Therefore all that is required to detect a collision is to calculate of the z-component: $a_1 b_2 - a_2 b_1$. If for any $\overline{Pv_n}$, $\overline{v_n v_{n+1}}$ pair this z-component is not positive, then it can be immediately determined that no collision has occurred, but if it can be shown for all pairs, then a collision has occurred.

This idea is demonstrated in Figure 5.1 in which a point P is being tested against a polygon ABCD. Calculation of the z-component of the cross product for this example reveals that $\overline{PAB}$ is negative, and $\overline{PBC}$, $\overline{PCD}$ and $\overline{PDA}$ are all positive. Because one of the z-components is negative, then the point must be outside the obstacle.

5.7.2 Minima Detection

Local minima in the APF may reveal themselves at the robot level by the robot stopping completely at a point because resultant forces from the attractive and repulsive potentials cancel each other out. Alternatively, the robot

may oscillate around a position, moving towards the goal for a couple of steps only to be repelled back in subsequent steps. Both cases can be caught by keeping a buffer of the last few robot positions. A quick check to see if the current position has been visited in the 'recently visited buffer' provides a simple way to detect the presence of a minima at the robot level, rather than by evaluating the whole potential field, and seeking minima or saddle points in that way.

5.8 Graphical User Interface

The output from the GA is a tuple of numbers representing the optimised parameters. They are fed into the simulation together with the problem specification and a description of the workspace and consequently a path is output in the form of an ordered list of coordinate pairs which represent a path. The output from the simulation needs be presented in a way which is readily understandable, and from which insight can be gained as to how the algorithm under test is performing, to allow for the incremental improvement of the algorithm. A list of numbers does not offer a means of providing this.

A graphical user interface (GUI) allows easy visualisation of a problem. A workspace can be created and filled with obstacles, and an initial and goal position for a specific problem can be specified. The path resulting from the application of GA can be seen as a line, rather than as a list of numbers. The GUI used was a simple motif application which acted as one element of the overall GA planning package. Screen shots later in this chapter show the GUI in action[1].

Communication between the GA planner and the GUI was achieved through a series of files:

[1]The GUI was developed in coordination with a student visiting Brunel during an overseas placement [Veronneau, 1998]

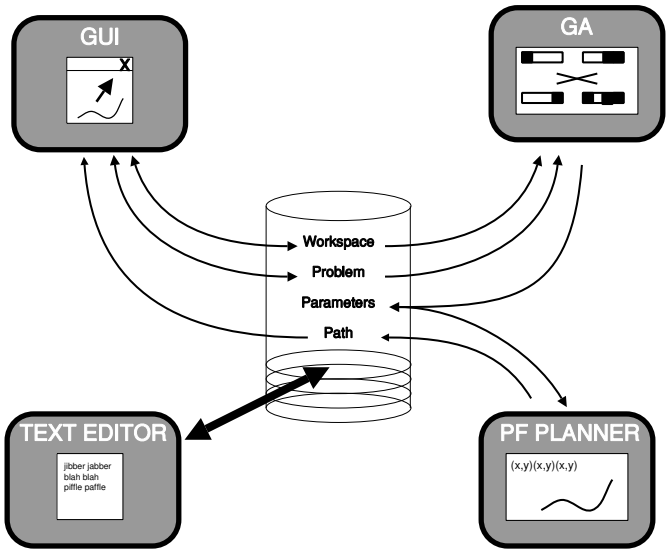


Figure 5.2: Interconnectivity of components of the GA APF planning system

Workspace Describing the workspace dimensions, and the obstacles within it.

Problem Describing the initial and goal position.

Parameters Detailing the APF function parameters and sub-goal positions evolved by the GA.

Path Ordered list of coordinate pairs describing the path.

The framework shown in Figure 5.2 allows software components to be easily changed. For example, much of the introductory coding was carried out without the aid of the GUI which was slotted into the framework later in the project without any change to existing components. The contents of each of the files can be edited manually, or can be created automatically by one of the software components. This framework proved to be an effective tool for carrying out the experimentation for this work.

5.9 Fitness

As with all genetic applications, the purpose of the fitness is to provide an indication of how good a particular evolved solution is at doing its job. For

this problem, the aim is for the evolved parameters to optimise the potential field function in order to create an almost global navigation function.

This application of GA does not try to approximate an input data set — there is no 'perfect' potential field which is being aimed at. The very essence of the problem is that such a field is difficult to define manually, which is the very reason for using GA. The only 'data' available is the fact the robot should move from the start to the goal. A reinforcement learning approach is used whereby a solution which offers a good solution to this requirement is rewarded, and one which performs badly is punished. The difficulty is in defining a fitness measure which accurately captures the requirements of the problem.

Once again in this application of GA, the problem of a multi-faceted objective function is encountered. Although the primary aim is to reach the goal position, there is also a requirement that the path followed should be as short as possible, and that the robot should not collide with obstacles en-route. These goals can conflict, so what is required is a method for finding the best trade-off between each of these goals. In Chapter 4, an approach was adopted whereby the secondary fitness objectives are scaled by a function of the primary fitness. A variant of this approach adapted for the APF optimisation problem is used here.

A desirable fitness function returns monotonically increasing values representing the abilities falling between the two extremes of perfect and imperfect individual.

The elements which comprise the fitness function are shown in Table 5.1. If weights are to be applied to various elements of the fitness, it is important that the weight only adjusts the significance placed on that fitness objective, and is not simultaneously performing normalisation. For this reason, each of the variables are normalised before combination into the multi-objective fitness function, so that their values fall into the range 0.0-1.0. This allows competition between the fitness objectives at an equal level. The normalised

forms are subscripted with an n in Table 5.1, and maximum values are subscripted with *max*.

Symbol	Description	Definition (if any)
w	Weighting factor	$\begin{cases} 0.3 & if\ s_n = 0 \\ 0.3 \times (1.0 - d_n) & if\ s_n > 0 \end{cases}$
s_n	Number of Steps	s/s_{max}
d_n	Euclidean distance between rest and goal	$d/\sqrt{2}$
c	Boolean style collision indicator	$\begin{cases} 1.0 & if\ collision\ occurred \\ 0.0 & if\ no\ collision\ occurred \end{cases}$
m	Boolean style minima indicator	$\begin{cases} 1.0 & if\ minima\ encountered \\ 0.0 & if\ no\ minima\ encountered \end{cases}$
l	Number of intersections on the line between the resting and goal positions, with obstacle boundaries.	

Table 5.1: Components of the multi-objective GA APF fitness function

The multi-objective fitness function is defined as:

$$(0.3w(s_n + c + m) + (1 - w)d_n) * l$$

The adaptive weight w is used to split the weighting between the primary goal of moving nearer to the goal, and minimising the number of moves and collisions. The constant 0.3 means that the secondary fitness objectives can never contribute to more that 30% of the fitness value. This is an arbitrary value which was found to work well in general situations and for different environments. If the weighting of the secondary fitness objectives was allowed

to reach too high, the performance of the best individual in the evolved population tended to converge too early.

The downside of the split adaptive weight is that the improvement of the population cannot be monitored using the multi-objective fitness. The fitness function is not static, outputting absolute values, rather it is a dynamic function outputting relative values. A demonstration of the effect of the use of this adaptive weighting technique is given when used to solve a more difficult path planning problem in Section 5.13.

The primary objective of arriving at the goal has to be measured in terms of some sort of distance. Discussions have already been made in Chapter 4 as to the undesirable nature of Euclidean distance as a measure for path planning problems, and equally the difficulty in generating a true distance-to-goal value as this requires the solution to be known in advance. The approximation here uses the Euclidean distance, multiplied by the number of times the line of sight between the resting and goal position is broken by obstacle boundaries. This not perfect, but does encourage solutions to reduce the number of obstacles between the resting position and the goal to a minimum.

5.10 First Steps in the Application of GA APF path planning

Test Case 1

Initial attempts at path planning were made using an environment containing three square obstacles. The tableau for this problem is shown in Table 5.2.

The initial and goal positions were placed diagonally opposite at $(10, 10)$ and $(90, 90)$ respectively as shown in Figure 5.3. The environment was designed so as to place three obstacles in the direct line-of-sight between the initial and

Objective:	Test Case 1: Evolve a successful plan from initial position=(10,10) to goal position=(90,90) in Workspace (Figure 5.3)
Evolved Parameters	Goal(1): ξ; Obstacles(3) ρ_0, η; Subgoals(0)
Fitness:	Adaptive weighted multi-objective function (Section 5.9)
GA Parameters:	population = 1000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.2: GA Tableau for Test Case 1

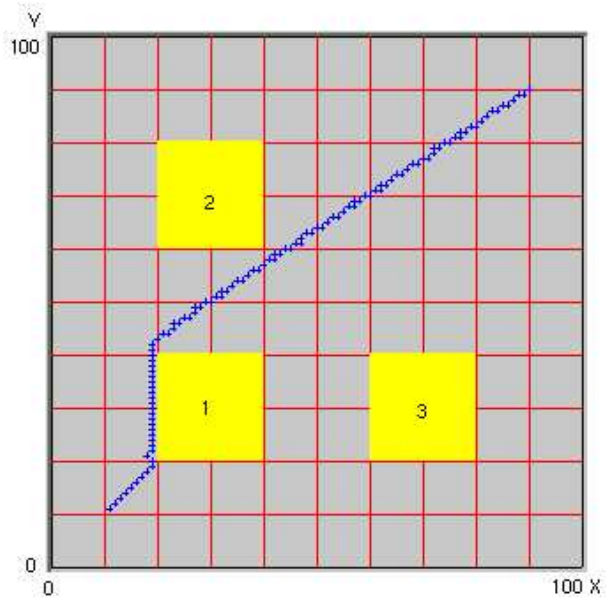


Figure 5.3: Test Case 1: Simple, three obstacle environment with start=$(10,10)$, goal=$(90,90)$

goal positions. It was intended to highlight any major flaws in the approach before more difficult scenarios were attempted.

The planner is able to successfully plan a path of length 151 for this problem, evolving parameters which when instantiated in the APF functions described earlier give rise to the potential field shown in Figure 5.4. The evolved parameters are shown in Table 5.3.

The plot in Figure 5.4(a) shows a 'landscape' on which it may be envisaged

Goal	obstacles					
	1		1		2	
ξ	ρ_0	η	ρ_0	η	ρ_0	η
90.6248	0.0061	0.7073	25.3895	0.0095	0.0237	0.6301

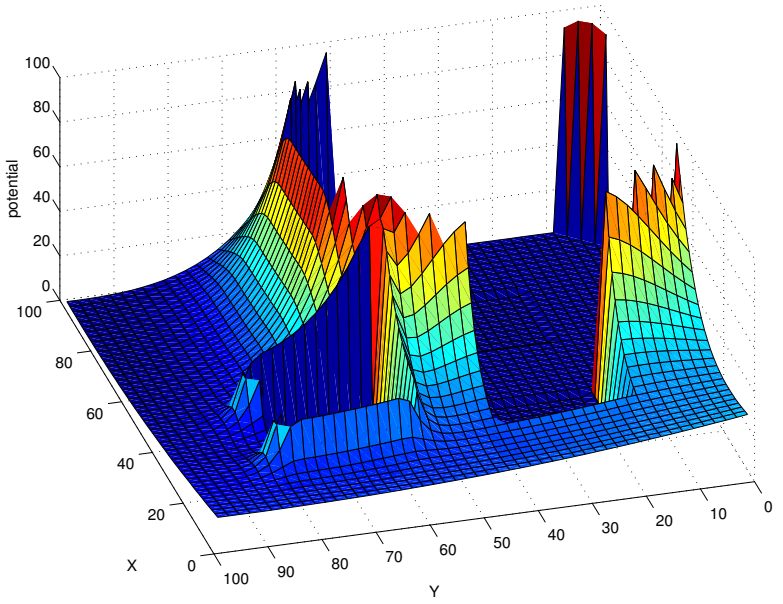
Table 5.3: Successful evolved parameters for Test Case 1

that a marble could be placed and would roll to the goal. The quiver plot in Figure 5.4(b) shows the direction which a robot would move at any point in order to move towards the goal. To generate these plots, evaluation of the APF functions was obviously required for every position in the environment, however in generating a plan, this is not required — only positions on, and adjacent to the path need be evaluated.

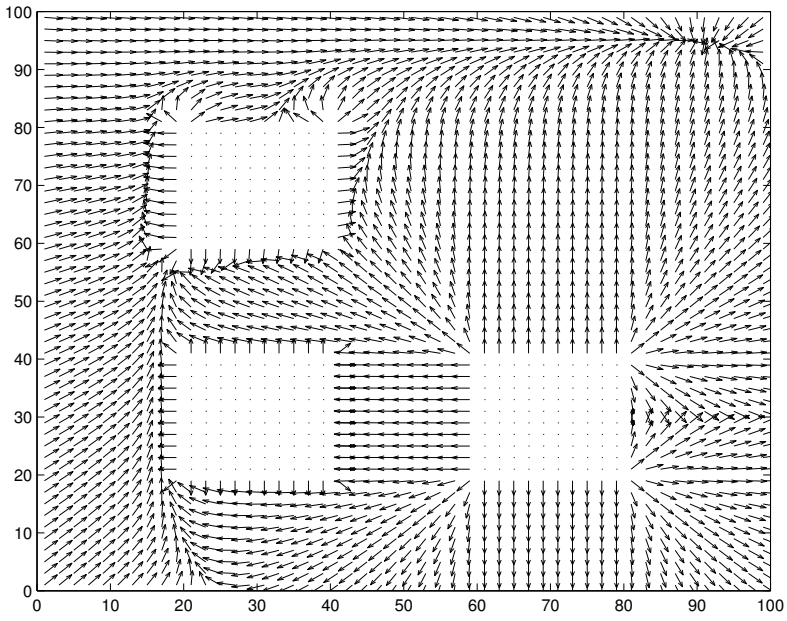
As is demonstrated clearly on the Quiver Plot in Figure 5.4(b), the evolved plot allows for generalisation over different initial positions, that is with a fixed goal position. It is not possible to generalise over both initial and goal positions as this is akin to summing and averaging all the potential fields for all possible goal positions. To perform path planning with APFs, there must be a global minima. By definition, it is not possible to have one potential field with many global minima. It has been mentioned that numerical potential functions can successfully account for multiple goals, however, not for all goal positions. The 'landscape' would simply be flat with no gradient to follow — if all positions in the environment are goals, then in all cases the start and goal positions are in the same place.

5.11 Comparisons with a Manual Human Attempt

It was shown in the previous experiment that the GA is able to optimise seven parameters to allow a successful path to be generated. What is not clear, however, is how this GA approach compares with manual tuning of the results. On first glance it was not expected that the task would be too difficult, but as Table 5.4 shows, this is not the case. Each subject was presented with a text file containing zero valued parameters. Subjects were purposefully chosen from



(a)



(b)

Figure 5.4: APF for Test Case 1 shown as (a) a Surface Plot, and (b) a Quiver Plot

Subject	time (minutes)	distance to goal (as crow files)	moves
1	24	78	74
2	32	78	74
3	50	38	120
4	50	55	101
GA (first)	2	0	180
GA (refined)	12	0	122

Table 5.4: Results of manual attempts at tuning seven APF function parameters

the PhD student community who had an affinity with technical computing, and were aware of the line of research being undertaken in this dissertation. The purpose of each of the parameters was explained to them, as well as the range within which each could fall. They were also given the ability to view the path on the GUI, and produce a surface or quiver plot of the APF corresponding to the parameter values which they had set.

After an initial period of 'playing' with the parameters for about 10 minutes, the subjects deduced that another force was required other than the attractive goal force and repulsive force from obstacle 1. These two forces alone would always results in a path which had points only on the diagonal between the initial and goal position. Another initial force was required which provided the initial momentum to start the robot on its journey. To achieve this, the subjects would increase the size of the distance of influence around either obstacle 2 or 3, so that the initial position was under the influence of another repulsive force. This typically allowed the path to suddenly grow from around 2 moves to around 40 or 50. Progress then slowed, as it took longer and longer to adjust the parameters, and cope with the interdependencies of the parameters. Although their paths were following similar directions as the GA generated paths, no human subject was able to produce a path which was

nearer than around 0.3 to 0.4 of the workspace, and around 96 moves.

The graph in Figure 5.5 shows the progress of manual attempts to set the parameters for this problem. The graph shows how the number of moves made by the robot increased as the parameters were tuned over time. For comparison, the figures for the GA run were also plotted. It must be noted that the GA found a solution in less than two minutes, after which the number of moves decreased as the path was refined and made shorter. one of the human subjects managed to complete a successful path.

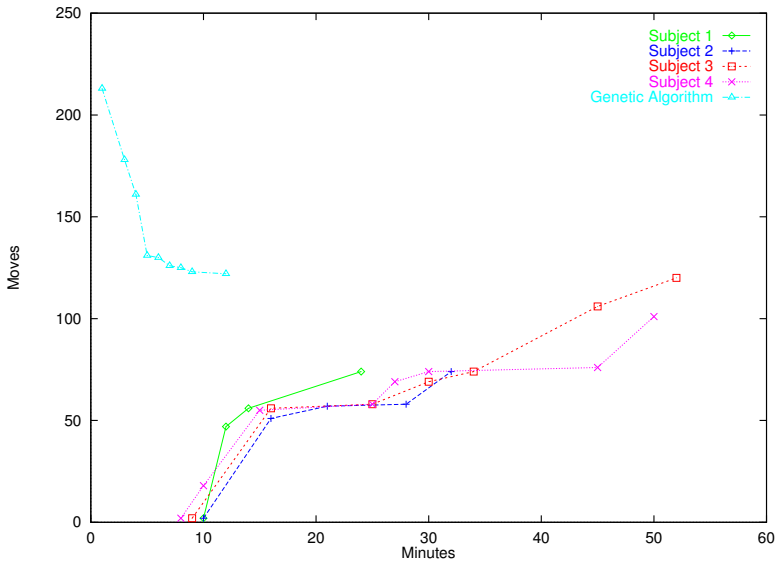


Figure 5.5: Progress of attempts to manually set APF parameters for Test Case 1

5.12 The use of subgoals

Test Case 2

The simple three obstacle environment demonstrated the limited ability of an APF planner. The scenario presented in Test Case 2 (Figure 5.6) introduces a major problem for an APF planner. The tableau for this problem is shown in Table 5.5.

Objective:	Test Case 1: Evolve a successful plan from initial position=(50,65) to goal position=(50,80) in Workspace (Figure 5.6)
Evolved Parameters	Goal(1): ξ; Obstacles(1) ρ_0, η; Subgoals(1): σ, α
Fitness:	Adaptive weighted multi-objective function for subgoals (Section 5.12.2)
GA Parameters:	population = 1000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.5: GA Tableau for Test Case 2

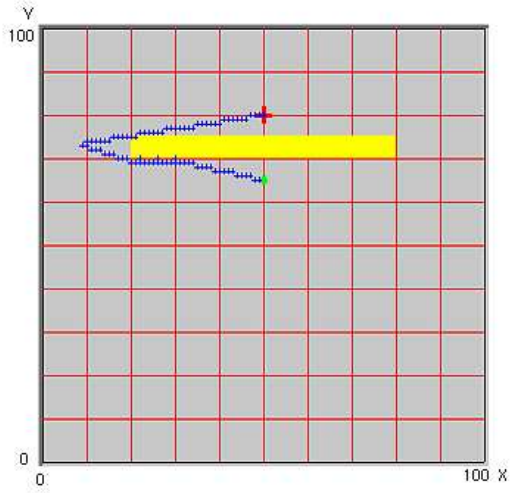


Figure 5.6: Test Case 2: Path Planning problem with inevitable local minima

This problem is difficult not simply because there is an obstacle in the line of sight between the initial and goal positions, but because the initial and goal share an x-coordinate, and the obstacle facing the initial position is perpendicular to the line-of-sight. The result of these circumstances is that it is guaranteed that the robot will hit a local minima. Because the initial position is directly in line with the goal, there is no lateral attractive force, and similarly, because the facing edge of the obstacle is perpendicular to the path of the robot, there are no lateral repulsive forces emanating from the obstacle. This lack of lateral forces means that the robot is unable to move round the obstacle. The only way this is possible, is if there is another attractive

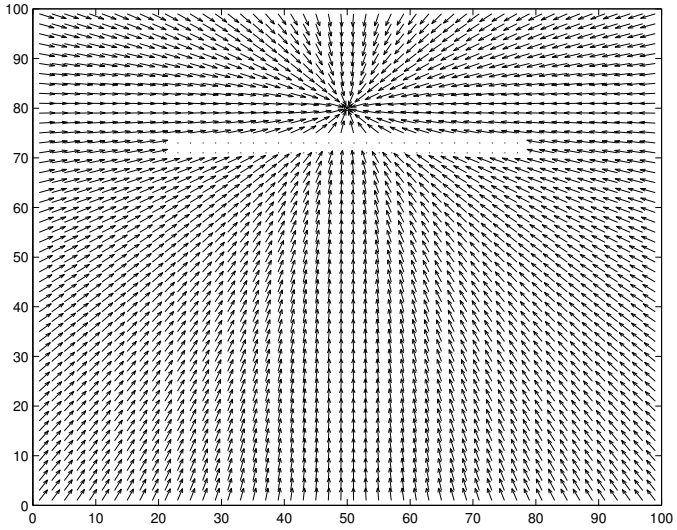


Figure 5.7: Quiver Plot for Test Case 2 with inevitable local minima.

force to pull it sideways. The GA is able to position a sub-goal, and set the APF parameters to solve this problem. The evolved parameters are shown in Table 5.6.

Goal	obstacle		sub-goal		
ξ	ρ_0	η	(x,y)	σ	α
70.0298	14.5571	0.0049	(10,77)	3.5370	0.0599

Table 5.6: Successful evolved parameters for Test Case 2

The case where the obstacle is directly in line with the goal is the extreme case, but the wide obstacle means that a 'shadow' is cast over a large area of the workspace, causing paths entering this region to be pulled towards the local minima in line with the goal, but on the opposite side of the obstacle. This can be seen by the quiver plot in Figure 5.7. Many of the arrows below the obstacle are orientated inwards, pointing the way to a local minimum.

This example scenario highlights a problem which requires a means of introducing another force into the workspace to supply some lateral force to allow the path to move round the obstacle rather than attempting to pass directly through it. To this end, one or more subgoals were introduced to the environment.

5.12.1 Implementation of Subgoals

Adding subgoals to the GA representation requires the addition of a tuple to the GA string for each subgoal used. As many subgoals as required can be catenated on the end of the string. Initially it was thought that a subgoal could be defined using an (x,y) coordinate pair to position it in the environment, and a gain value, equivalent to ξ which is referred to as σ. However, the simultaneous presence of more than one goal introduces a new problem as it removes another. Considering the environment in Figure 5.6, if a subgoal is placed to the left or right of the obstacle to pull the robot around the obstacle, then inevitably, the attractive force of the real goal will not be strong enough to pull the robot back towards it once the robot has cleared the obstacle. All that is achieved by introducing a simultaneously active subgoal, is that the global minimum in the landscape is moved away from the ultimate goal.

The solution to this problem is to allow the GA to evolve a sequence of subgoals, only one of which is active at any one time. The order in which they are activated is determined by the GA. This is achieved by making the first subgoal in the string active first, followed by the second, and so on. The final goal to be active is, of course, the ultimate goal. The mechanism to switch from one goal to the next can be implemented by introducing a deactivation distance α around each obstacle. This is similar to the distance of influence surrounding each obstacle. When the robot path enters the deactivation area, the next ordered subgoal becomes active. The importance of order in the GA representation is discussed in more detail by Davidor [1989].

In introducing subgoals, the GA string will consist of one parameter for the ultimate goal, two parameters for each obstacle in the workspace, and a further four parameters for each subgoal used. These four parameters are the (x,y) coordinate pair to position the sub-goal, the gain of the attractive well associated with the sub-goal, and the trigger radius around the goal, which when breached by the robot will make the next goal the active one.

5.12.2 Subgoal Fitness

The introduction of subgoals into the APF planner brings with it the requirement to adjust the fitness, to encourage the best possible configuration of the subgoals. Table 5.7 shows the additional fitness elements which are introduced.

Symbol	Description	Definition (if any)
g_n	Cumulative normalised distance between subgoals	$\sum_{i=1}^{subgoals} \frac{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}}{\sqrt{2}}$
a	Active Goal	a is currently active subgoal, ranging from 0—t, where t is number of subgoals, and the first subgoal, and 0 is the ultimate goal.

Table 5.7: Fitness components to account for the use of subgoals in the multi-objective GA APF fitness function

These additional fitness measures are added to the fitness function:

$$(0.3w(s_n + c + m) + (1 - w)(d_n + g_n)) * a$$

The cumulative distance provides a coarse granularity measure of the path length, and hence the lower the g_n, the shorter the robot path. The active goal measurement is to encourage all the subgoals to be used. It is unlikely that a path will reach the ultimate goal unless all of the subgoals have fired in turn. By multiplying the entire fitness by a, a heavy penalty is imposed for unused subgoals. In fact, it was found that it was not necessary to use g_n — it did contribute to the fitness, but removing it did not make a noticeable difference.

On one hand the introduction of sub-goals allows otherwise unsolvable problems to be solved, and secondly it can improve the quality of solutions to some

problems by breaking them down into simpler constituent problems. A downside of the use of subgoals, is that it is not possible to view a single diagram of an APF field, because there are as many fields as their are subgoals.

5.13 More Difficult Problems

Test Case 3

This section demonstrates the performance of the path planner against examples with various more complex environments. The tableau for this problem is shown in Table 5.8.

Objective:	Test Case 3: Evolve a successful plan from initial position=(10,10) to goal position=(50,80) in Workspace (Figure 5.8)
Evolved Parameters	Goal(1): ξ; Obstacles(2) ρ_0, η; Subgoals(4) σ, α
Fitness:	Adaptive weighted multi-objective function for subgoals (Section 5.12.2)
GA Parameters:	population = 5000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.8: GA Tableau for Test Case 3

Test Case 3 uses two horizontal obstacles between the initial and goal positions which span 80% of the room, and are placed above each other but slightly offset from each other (Figure 5.8). The only route the robot can take is a zig-zag pattern which is difficult to achieve, because with an attractive well situated at the goal, the robot tries to move towards the goal through the obstacles. The repulsive field surrounding an obstacle prevents the robot from colliding with it, but the adverse result of which is that the robot becomes stuck in a local minima. It is only through a combination of the multi-objective fitness function using adaptive weighting (Section 5.9), and the use of sub-goals that this problem could be solved. The progress of the run for

Test Case 3, shown in Figure 5.8 shows how initially a successful path was evolved, and then this path was refined to make it shorter.

The parameters evolved at generation 67 are shown in Table 5.9..

Goal	obstacles			
	1		2	
ξ	ρ_0	η	ρ_0	η
20.53	20.06	0.0057	15.31	0.0158

sub-goals											
1			2			3			4		
(x,y)	σ	α	(x,y)	σ	α	(x,y)	σ	α	(x,y)	σ	α
(16,85)	5.740	0.259	(7,40)	0.024	0.039	(84,37)	8.230	0.065	(89,18)	15.973	0.075

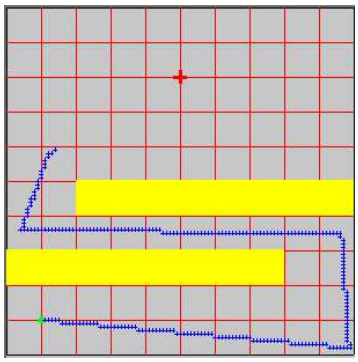
Table 5.9: Successful evolved parameters for Test Case 3

The graphs in Figure 5.9 correspond to the Test Case 3 run, and show how the distance to the goal, and the number of moves made by the best individual at each generation changed during the course of the run shown in Figure 5.8.

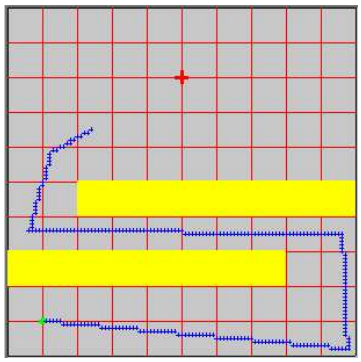
Because of the adaptive weighting built into the fitness function, the emphasis placed on minimising the number of moves is low at the start of the run, the distance to the goal reduces, and the number of moves made to achieve the path is allowed to increase. This increase in moves is obviously necessary to allow the robot to move closer to the goal. The primary concern of the GA at the beginning of the run is to move closer to the goal.

At generation 19, there is a simultaneously peaking of the number of moves, and minimisation of the distance to goal, as the path arrives at the goal. During these early generations, the weights ratio of distance to goal:number of moves gradually changed from an 83:17 to 70:30. The remainder of the run maintains the primary objective of minimising the distance to the goal, whilst reducing the number of moves taken to achieve that successful path from 298 to 226.

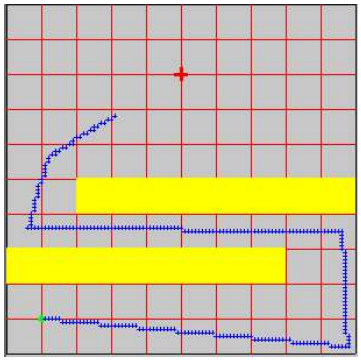
The progress of the path through the generations of a successful run is shown in Figure 5.8. The top four examples show the path moving closer to, and finally reaching the goal, and the bottom four show how, having reached the goal, the path is refined and gradually made shorter.



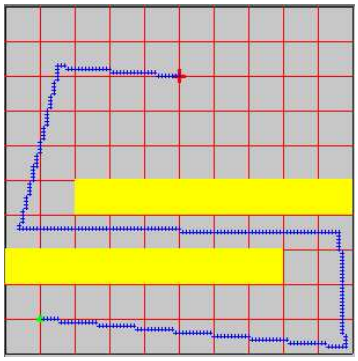
Generation 12



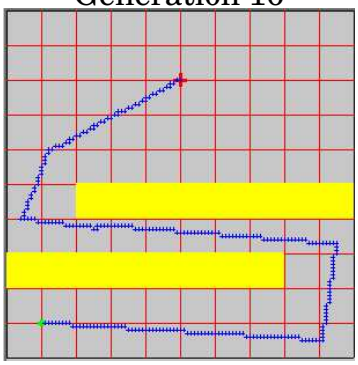
Generation 15



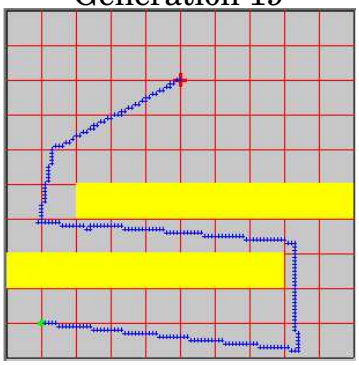
Generation 16



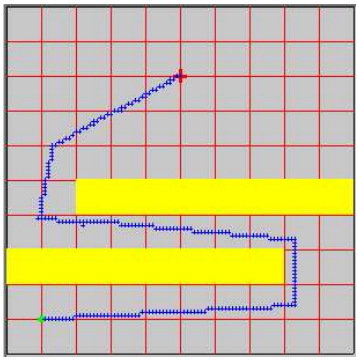
Generation 19



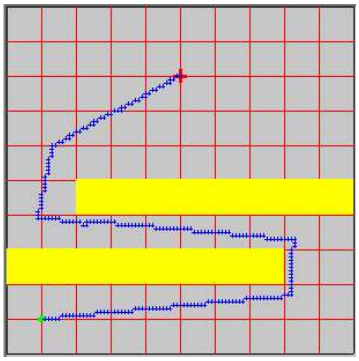
Generation 28



Generation 35

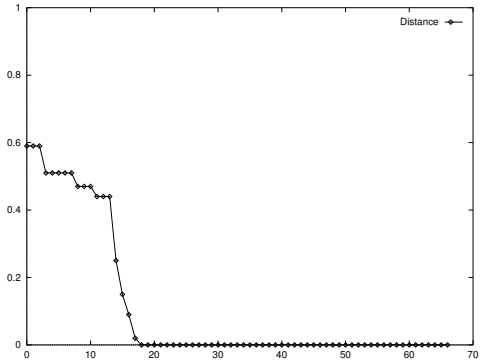


Generation 53

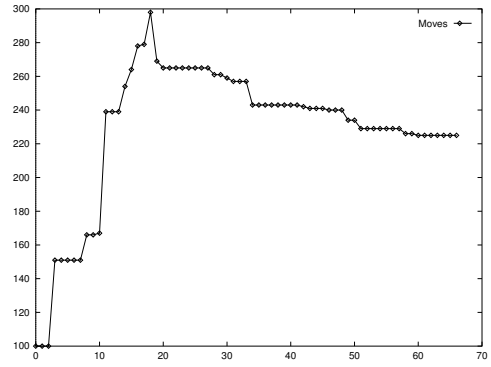


Generation 67

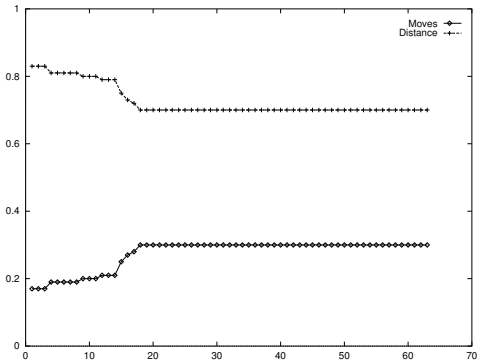
Figure 5.8: Test Case 3: Demonstration of the refinement of a successful path.



Distance between rest and goal



Moves



Weights

Figure 5.9: Graphs of distance between goal and resting position, number of moves in path and weighting emphasis for Test Case 3

Test Case 4

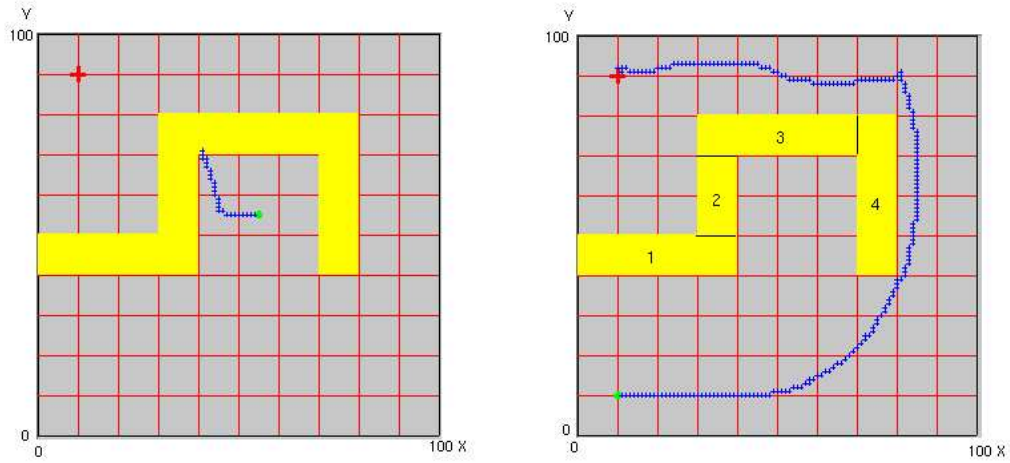
Test Case 4 represents a more difficult problem for the planner, where the initial and goal position are relatively close to one another, but they are divided by a trap which requires quite a lot of moves to negotiate. The tableau for the problem is shown in Table 5.10.

Figure 5.10 shows two scenarios with a new workspace, one in which a plan is successfully generated (b) and the other in which the planner has failed (a).

When trying to plan from $(55,55)$ to $(10,90)$, the best path, as shown in Figure 5.10(a) is one which simply attempts to burrow through the wall, directly towards the goal. The close proximity of the goal to the initial position, and the positioning of the obstacles around the initial position mean that it is dif-

Objective:	Test Case 4: Evolve a successful plan from (a) initial position=(55,55) to goal position=(10,90) (b) initial position=(15,55) to goal position=(10,90) in Workspace (Figure 5.10)
Evolved Parameters	Goal(1): ξ; Obstacles(4) ρ_0, η; Subgoals((a)-0, (b)-3): σ, α
Fitness:	Adaptive weighted multi-objective function for subgoals (Section 5.12.2)
GA Parameters:	population = 1000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.10: GA Tableau for Test Case 4



(a) Failure - init = $(55,55)$, goal = $(10,90)$ (no subgoals)

(b) Success - init = $(10,10)$, goal = $(10,90)$ (3 subgoals)

Figure 5.10: Test Case 4: Successful and Unsuccessful plans

ficult to encourage the GA to explore the workspace. It is content with the 'good enough' fitness which it achieves by moving to the interceding wall, and no further. The GA population is quickly polluted with 'bee-line' individuals which exclude the possibility for the evolution of better solutions. Using larger populations only serves to increase run times. Although Figure 5.10(a) is taken from a run with no subgoals, even when subgoals are used the GA does not have enough incentive to search around for better alternatives.

A successful plan can be generated for the scenario shown in Figure 5.10(b)

where the aim is to move from $(10,10)$ to $(10,90)$. Because the initial position is further away from the goal, evolved sets of parameters which result in a 'bee-line' do not pollute the population with their sub-optimal 'genes', allowing the GA to explore, and eventually arrive at a good solution. The successful path, evolved at generation 23, is 225 moves long. The evolved parameters for the successful path in Figure 5.10(b) are shown in Table 5.11.

Goal	obstacles							
	1		2		3		4	
ξ	ρ_0	η	ρ_0	η	ρ_0	η	ρ_0	η
38.687725	2.916	0.453	25.415	0.216	36.051	0.097	36.895	0.005

sub-goals								
1			2			3		
(x,y)	σ	α	(x,y)	σ	α	(x,y)	σ	α
(91,94)	31.245	0.082	(76,61)	13.552	0.961	(97,43)	35.195	0.309

Table 5.11: Successful evolved parameters for Test Case 4

Test Case 4 demonstrates how important the fitness measure is. The adaptively weighted, multi-objective fitness function developed for this planner has shown itself to be very successful. However, the fact that it cannot perform for all scenarios indicates the need for further fitness function development, and for some kind of fail-safe backup planner for cases when the GA/APF planner does not produce a successful plan.

Test Case 5

This experiment uses a room which is more heavily cluttered with five obstacles. There is, therefore, less free space in which the robot can move. The tableau for this problem is shown in Table 5.12.

A successful path was evolved at generation 76 with 147 moves. This is shown in Figure 5.11. The evolved parameters for this successful plan are shown in Table 5.13.

Objective:	Test Case 5: Evolve a successful plan from initial position=(10,10) to goal position=(90,90) in Workspace (Figure 5.11)
Evolved Parameters	Goal(1): ξ; Obstacles(5) ρ_0, η; Subgoals(4): σ, α
Fitness:	Adaptive weighted multi-objective function for subgoals (Section 5.12.2)
GA Parameters:	population = 4000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.12: GA Tableau for Test Case 5

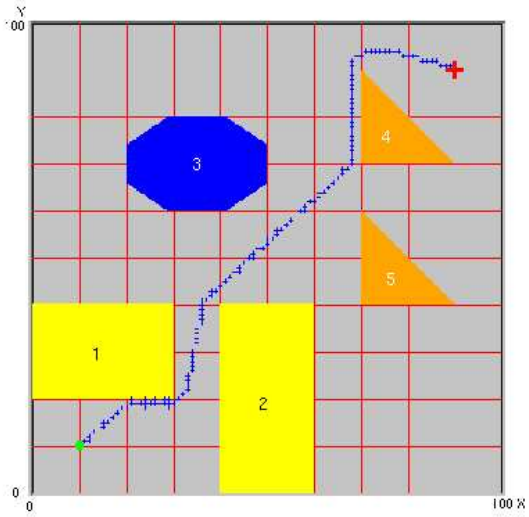


Figure 5.11: Successful path for Test Case 5: a 100×100 workspace cluttered with five miscellaneous obstacles

Test Case 6

It has been mentioned previously that elliptical obstacles are easier to plan for than rectangular ones. This is because they do not have the sharp changes

Goal	obstacles									
	1		2		3		4		5	
ξ	ρ_0	η	ρ_0	η	ρ_0	η	ρ_0	η	ρ_0	η
86.154	37.576	0.008	16.296	0.016	15.166	0.047	0.008	0.803	49.703	0.030

sub-goals											
1			2			3			4		
(x,y)	σ	α	(x,y)	σ	α	(x,y)	σ	α	(x,y)	σ	α
(67,83)	47.475	0.758	(40,68)	25.497	0.963	(48,99)	48.437	0.620	(73,64)	37.976	0.599

Table 5.13: Successful evolved parameters for Test Case 5

in direction of repulsive force associated with the vertices which may encourage local minima. Using a polygonal representation, where all obstacles are constructed from straight line segments, it is not possible to define an ellipse, however an approximation can be made, in the case of this experiment, an octagon is used. The tableau for Test Case 6 is shown in Table 5.14.

Objective:	Test Case 6: Evolve a successful plan from initial position=(50,150) to goal position=(135,30) in Workspace (Figure 5.12)
Evolved Parameters	Goal(1): ξ; Obstacles(8) ρ_0, η
Fitness:	Adaptive weighted multi-objective function for subgoals (Section 5.9)
GA Parameters:	population = 6000, generations = 100, crossover = 0.70, reproduction = 0.28, mutation = 0.02
Success predicate:	Distance to goal = 0

Table 5.14: GA Tableau for Test Case 6

Figure 5.11 shows a larger 200×200 workspace, which is more heavily cluttered with octagonal obstacles. A successful solution, of 78 steps can be generated for this workspace without the use of any subgoals, however a larger population was required to account for the larger number of obstacles present in the workspace.

Goal	obstacles							
	1		2		3		4	
ξ	ρ_0	η	ρ_0	η	ρ_0	η	ρ_0	η
7.698	24.846	0.014	39.247	0.001	32.274	0.028	30.679	0.015

obstacles							
5		6		7		8	
ρ_0	η	ρ_0	η	ρ_0	η	ρ_0	η
0.690	0.036	41.695	0.548	12.019	0.073	5.895	0.068

Table 5.15: Successful evolved parameters for Test Case 6

5.14 Summary

This chapter has explained how it is possible to use GA to optimise an pre-existing technique used in path planning. The literature in the APF area

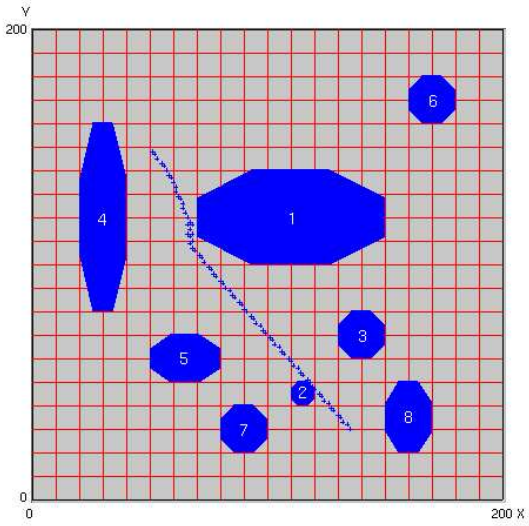


Figure 5.12: Successful path for Test Case 6: a 200×200 workspace cluttered with eight octagonal obstacles

offered few suggestions as to what successful methods could be employed to set the controlling parameters of Khatib's [1986] functions. Tests showed that manual optimal setting of these parameters is not easy, and using a GA to perform the optimisation is very successful.

As in the previous chapter, multi-objective fitness functions have played a central role in the success of the application of GA. The use of adaptive weights to balance the importance placed on primary and secondary fitness targets gave rise to successful solutions.

At this point in the dissertation, a number of new approaches to path planning have been offered. This and the previous chapter have evaluated individual examples of path planning problems tackled with evolutionary approaches. However it is now important to evaluate the more general adoption of GA and GP in path planning and look to how this might be achieved in the future.

6

The Future of Evolutionary Path Planning

6.1 Introduction

At this point in this dissertation, various methods have been proposed and tested which in their own way tackle the path planning problem. Specific issues arising from the implementation of each of these different approaches have been addressed in the previous two chapters. In this chapter a more general evaluation is made.

At the beginning of this dissertation, the *utopian* ideal of a fully independent robot was discussed. The production of such a robot is outside the scope of this thesis which has focused on path-planning; a very specific, and very important, component of such a robot. However, when focusing on the specific, it is worthwhile remaining aware of the greater goals. This chapter describes the successes of the techniques developed in Chapters 4 and 5, and also considers how the techniques which have arisen from this research might contribute to the future of path planning and robotics in terms of how it might be integrated with other technologies, and how it might form the seed for future research.

Consideration is given to how the approaches developed here might fit into a more complete motion planning system, and in particular how the GA optimisation of an APF for path planning might be combined with other technologies. Issues of performance in a real-time environment, and generalisation

are discussed with a view to the possible future realisation of a robot control system based on Evolutionary path planning.

6.2 Review of Proposed Evolutionary Path Planning Approaches

Chapters 4 and 5 documented the development of a range of different approaches which sought to apply GA and GP to the path planning problem. Earlier chapters had paved the way for this experimental work by identifying a need for improved path planning techniques; by providing useful background knowledge as to what shortcomings were found in existing techniques; and by introducing GA an GP as a potential means for improving path planning.

The preliminary investigations revealed a number of important points which were considered during the development of evolutionary path planning approaches, built on the underlying consideration that a useful path planning technique must eventually be practical, for example in terms of cost and speed of planning required:

Generalisation	The ability for a planner to be trained to generate plans for previously unseen situations.
Speed	The ability to plan in a timely manner.
Workspace Independence	The ability of a technique to be applied to problems of greater resolution, size of workspace, number of obstacles without incurring significantly greater resource requirement.

The fact that a successful planner must combine all these points, and of course generate successful paths, is the very reason that the development of such a

planner continues to elude researchers. There is no pretence that the application examples in Chapters 4 and 5 satisfy all these points, indeed it would have been extremely optimistic to assume that a single PhD dissertation would have achieved this. However, significant contribution is made by both the GP and GA approaches. They demonstrate successful examples of evolutionary path planning and and provide valuable input for those continuing to research along the path which will eventually lead to even better path planners.

6.2.1 Evolving a planner with GP

The use of GP to evolve a planner, rather than simply a one-off plan, particularly addressed the generalisation issue. The evolved planner, although possibly taking a while to create, would run quickly online, rather like the oral cancer diagnosis tool (Appendix A) which ran in a fraction of a second. The work acts somewhat as a stress test, defining limits as to what GP is capable of given the current maturity of the field, and the current availability of hardware. No use was made of special operators specific to the problem, such as were used by Xiao et al. [1997] in the GA path planning, indeed the reliance on domain knowledge built in by the programmer was avoided as far as possible. An independent robot cannot afford the luxury of a human nursemaid to guide it at all times if it is to be used in dangerous, or inaccessible environments.

Using GP to evolve a planner is a very difficult problem. In nature, evidence exists of the evolution of enormously complex systems being evolved which represent solutions, of a sort, to the problems of each animal living in their respective environments. In GA and GP, an attempt to mimic the evolutionary process is made, but even the largest GP runs, such as those undertaken by Andre and Koza [1996] and Bennett III et al. [1997] are miniscule when compared with the evolutionary process in nature which has given rise to modern

species. Natural evolution has been occurring for several billion years. Man, the most complex solution to date, appeared 200 million years ago in an early form. The point being made is that in terms of current GP and GA runs, man has been evolving for very many generations on a very powerful computer with a lot of memory — one which is not available to us, at least at present.

6.2.2 Optimising Potential Field Planning with GA

Whilst the goal of generalisation remained a consideration, the work in Chapter 5 concentrated more on the issue of speed and workspace independence. The APF planning method was chosen because it offered a simple, fast algorithm. Equally, GA is generally faster than GP because it typically has a more compact data structure — this is certainly the case for this application.

Using GA to enhance APF planning was shown to be successful. Not only was GA able to perform better than humans in the task of optimising the APF parameters, but the technique was also shown to be capable of generating paths for a number of different environments.

As described in Chapter 2, there are methods already in place for helping to avoid the effects of local minima present in a potential field. The use of pre-existing techniques, such as the execution of Brownian Motions [Barraquand and Latombe, 1991] to jump out of minima, or the use of virtual springs [McLean and Cameron, 1996] are not made obsolescent by this dissertation. Indeed, by combining more elements with the GA APF planner, further improvements can undoubtedly be made to the planner, however this is a subject for future research. In providing an automatic means for optimising the APF functions, rather than relying on manual tuning, the aim has been to provide a better foundation from which all other APF techniques can benefit. The GA optimisation of APF functions is not limited to the original attractive well and FIRAS functions [Khatib, 1986], but should be equally applicable to any attractive or repulsive functions which have tunable parameters.

6.3 Considerations for use of Evolutionary Computing for Path Planning

Evolutionary computing and specifically GA and GP have provided the main tools which have been used to approach the path planning problem. They have been used both on their own, and to improve an existing, classical approach to path planning. This dissertation does not define the limit for the application of these techniques. Indeed, this section suggests ways in which EC may be used to further improve path planning.

6.3.1 multi-objective fitness

Much of the success of the approach developed in Chapter 5 can be attributed to the adaptive fitness weighting used to continually change the emphasis placed on the primary and secondary fitness objectives used to measure the performance of a candidate solution. The fitness measure used in Chapter 5 was specifically tailored for the path planning application, and it may be possible to further refine the adaptive weight approach and make it more general, so it can be easily applied to other problems involving multi-objective fitness measures. Improvements in coping with multi-objective fitness will hopefully lead to a corresponding improvement in the GA optimisation of APF path planning.

6.3.2 Improved repulsive APF functions

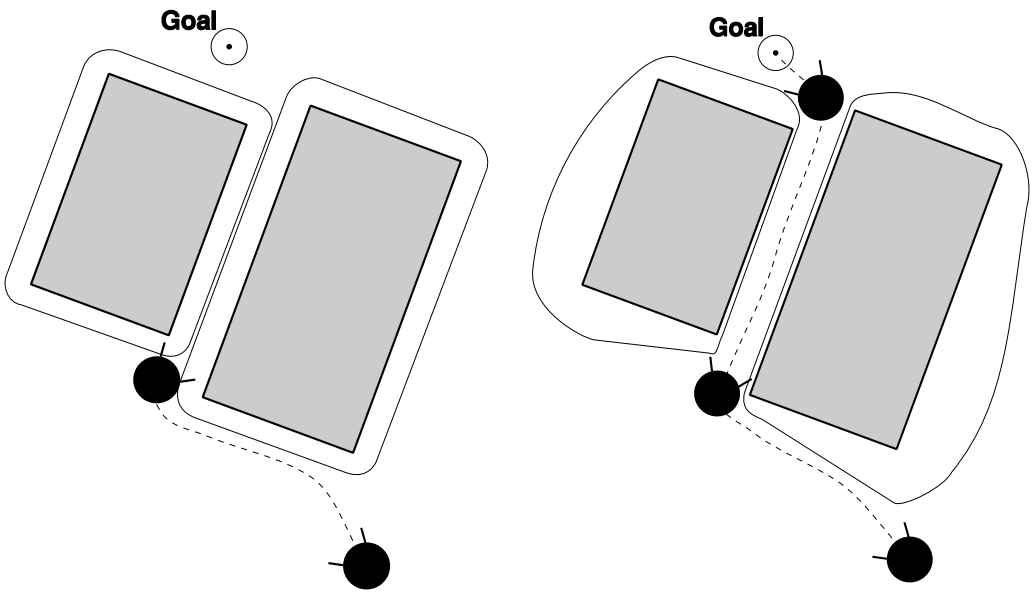
One component of the potential field used in APF planning is the attractive well used to encourage the robot to move towards the goal. However, the component which adds features corresponding to the obstacles in the workspace is the set of repulsive fields generated by the FIRAS function [Khatib, 1986]. When the GA is used to set the parameters for the potential field, most of

these parameters pertain to the FIRAS functions, and therefore the shape of the overall field is controlled by these parameters.

Results presented in this dissertation, and the work of others, for example Li et al. [1990] and Khatib [1986] show that the use of the FIRAS function as the repulsive function leads to fields from which successful plans can be generated. However, the FIRAS function generates a repulsive field which extends a uniform distance ρ_0 from the edges of the obstacle. If two obstacles are positioned close together, it may be desirable to have a very small value for ρ_0 such that the robot can pass between the two obstacles, if in doing so it can achieve a shorter path than by moving round the outside of the obstacles. On the other hand, on the edges of an obstacle which are not close to a neighbouring obstacle, it may be more desirable for the field to extend further from the edge to prevent the robot from coming unnecessarily close to the obstacle. Such a repulsive field cannot be generated using the FIRAS function. Figure 6.1 illustrates the scenarios described above. In Figure 6.1(a) two closely positioned obstacles have repulsive fields with uniform values for ρ_0 for all their edges. If this value of ρ_0 is sufficiently large such that the robot does not hug the obstacles, then the corridor between the obstacles which must be traversed to achieve the shortest path becomes blocked by repulsive fields. An alternative in Figure 6.1(b) shows a more desirable field where the robot is encouraged to keep clear of obstacles where it is appropriate to do so, but the value of ρ_0 may be set very small where that is the only way to permit the robot to pass through an area of the workspace.

Rather than using GA to optimise the FIRAS function, it may be possible to use GP to generate a non-uniform repulsive field for each obstacle from scratch. The GP representation scheme might consist of:

Function Set containing mathematical functions such as might be found in many GP function approximation problems.



(a) (b)

Figure 6.1: Illustration of a path plan between two closely positioned obstacles having a (a) uniform distance of influence, (b) non-uniform distance of influence

Terminal Set containing coordinates of the position of the robot and some random real number values.

The evolved mathematical function would output a potential value for a particular robot position. Because the function is evolved for a specific obstacle, the magnitude of the field and the distance of influence should be 'hard coded' into the function, and should not need to be supplied, for example, in the terminal set. An optimum field gain and distance of influence from each obstacle edge should be evolved by GP.

It is proposed that one GP process should be responsible for evolving the repulsive field function for a single obstacle. Typically, an environment will contain several obstacles, and correspondingly several GP processes will be required. In a multi-tasking environment, the GA process responsible for managing the attractive well, and the set of GP processes, each responsible for an obstacle in the workspace can all run simultaneously in a co-evolutionary approach [Juillé and Pollack, 1998; Ahluwalia and Fogarty, 1996].

A clear drawback to this suggested approach of combinations of multi-tasking GA and GP processes is the increase in execution time. It was noted that one of the benefits inherited from using GA rather than GP was the corresponding reduction in computation time required. The problem of evolving a function from scratch is easier to implement within a GP framework. The computation problem can hopefully be addressed by using a parallel approach, as has been done by others [Juille and Pollack, 1995; Andre and Koza, 1996] and an example of which was described in Section 3.9. The proposed framework fits naturally into a parallel environment, with one obstacle being dealt with by one processor with little inter-process communication required.

The precise formulation of the representation scheme, fitness function and parallel framework can only be speculated on here, however this seems to offer an interesting area for future work.

6.4 Generalisation

At the outset of this research the aim was to produce path planning algorithms which were trained against a set of examples, and which would then be able to generalise and work effectively against unseen examples.

Generalisation can take a long time to achieve in terms of the time it takes for GA or GP to learn a general, rather than a specific rule,. However the time consuming learning can be performed off-line, whilst the online execution of the rule can be performed relatively quickly.

Results from experiments with GP path planning in Chapter 4 did show that GP was able to successfully generalise results from training data to test data. The problem with the GP approaches was that the performance against seen data was not really high enough for practical use in a real path planning system. The problem was not so much with the ability to generalise, but with the ability of the representation scheme to allow evolved rules to be able to 'see' the workspace effectively.

When the move was made away from a pure GP approach to a hybrid GA/APF approach, there was an implicit requirement to relinquish a certain amount of ability to generalise. The APF planning method encourages a single minimum point in the potential field which is positioned at the goal. Using a perfect, single minima field, generalisation is achieved because having evolved a potential field on the basis of a set of seen initial positions, it should be possible to generate a path for other unseen initial positions. Whatever position is chosen as the starting point, there should always be a route downhill to the goal. By changing the definition of the APF approach slightly, the minimum could similarly be positioned at the initial position, and generalisation could be achieved over goals. However, it is not possible, using APF planning, to generalise over both initial and goal positions, because this would require a potential field with more than one global minimum, which is a contradictory requirement. So, although complete generalisation must be ruled out, an application which, for example has a recurring set of goal positions, but varying starting positions could benefit from the generalisation possible using APF planning.

When subgoals were introduced in Chapter 5 to improve the performance against certain more difficult path planning problems, the ability to generalise was further decreased. By introducing subgoals, path-planning problems were tackled by evolving a coarse grained path defined by the ordered list of sub-goals, each of which points had a corresponding potential field. Because the ordered list of sub-goals is always tailored to a particular initial/goal position pair, the ability to generalise is severely diminished, but at the expense of significantly improved path planning performance.

The experimental work for the thesis highlights the classical problem with learning algorithms whereby performance worsens, or training time increases, as the requirement for generalisation increases [Dracopoulos, 1997b; Werbos, 1998]. In order to improve the performance, the programmer gives the algorithm under development more and more 'help' by adding domain knowl-

edge, but inadvertently worsens the ability to generalise.

The GP work used a plain approach which did not adopt domain specific add-ons to improve performance. To produce a better planning, evolutionary algorithms, this time in the form of GA, were used with a known path-planning approach. To further improve performance sub-goals were added, and gradually the onus is moved from the learning algorithm to the programmer or researcher. The algorithm is necessarily left with less to learn in order to improve performance, but the ability to generalise is reduced because the learning algorithms influence on the final result is similarly diminished.

This dissertation has, therefore, demonstrated both that generalisation can be achieved, and that performance may be improved by adding various programmer influenced add-ons, but that the difficulty arises in balancing these two goals.

6.5 Real Time Evolutionary Path Planning

Experiments were carried out in Chapter 5 to provide a comparison between a human performing the optimisation of the APF parameters, compared to the computer using GA. This comparison showed that the optimisation problem is difficult, and that the GA is considerably faster and more able than humans. However, a human will not be performing the optimisation in an independent robot, so the fact that the computer is faster than a human is interesting, but it does not necessarily mean that the computer is fast enough.

In moving towards the fully independent automaton, time constraints become very important. When performing research in a virtual environment, as has been done in this research, although it may be desirable for the planning and execution of those plans to be quick, it does not really matter whether the robot takes 5 minutes or 5 days to complete its task — no detrimental result will occur in the real world.

To realise an independent robot, the inevitable transition must, at some point be made, from a virtual to the real world, and in the real world there exist real-time constraints. Real time constraints can be broadly broken down into soft and hard categories [Stallings, 1992, Chapter 6]. A soft real-time constraint exists when desirable time limits are imposed which a control system should meet, but if the limits are exceeded then no harm will result. In a hard real-time system, if the time limits are not met, then the system has failed. The failure may be measured, for example, by the number of people on the aircraft which crashed because the ground arrived too quickly.

The experiments carried out during this research did not execute quickly enough to be used in modern robotics applications. Most runs took in the order of minutes to run, rather than a fraction of a second which may typically be required in a real environment.

Another time related problem when using non-deterministic techniques such as GA or GP, is that there can never be a guarantee that results will be returned in a set time. If an algorithm has been developed to control a system, incorporating a set of controlling equations, it can be predicted how long it will take to execute the algorithm. A single run of GA or GP may find a solution in a few seconds, or it may never find a solution because it encounters a local minima in the search space.

These facts do not sound the death knell for GA and GP path planning. The future will undoubtedly bring speed improvements from the further development both of techniques underlying GA and GP, and of the hardware on which they are executed. Indeed the practical use of GA and GP has only been made possible by the relatively recent advances in hardware performance, and future advances should open more areas of application. The remainder of this section looks at two ways in which speedup might be achieved.

6.5.1 Parallel GA and GP

Chapter 3 discussed the use of parallel computing to improve the speed of execution. The exact method used in Chapter 3 may not be appropriate for use in a robotic environment, but it indicates that a multi-processor computer in a robot could speed up the use of GA or GP in path planning. A situation could be envisaged where a relatively large number of very simple processors could be combined to provide a significant speed up. During fitness evaluation for the APF approach, one processor could be used for each obstacle, thus eliminating the increase in computation corresponding to an increase in the complexity of the environment in which planning is performed.

6.5.2 Evolvable Hardware

Another area which has not been explored in this thesis, but which has shown promise in other areas of application is that of evolvable hardware. It might be very interesting to see how a planner could be evolved in a FPGA (Field Programmable Gate Array), in the same way as successful solutions have been evolved in other application areas [Thompson, 1996b]. Perhaps robot path planning will provide a suitable application for the CAM-brain [de Garis, 1994; Korkin et al., 1997; Nawa et al., 1998] which, although impressive in its potential, to date seems to be lacking in any significant application.

6.6 A Future Motion Planning System

Perhaps the ultimate aim in robotics is a fully independent robot. Such a robot may incorporate, amongst other technologies:

- **vision systems** to allow the robot to see the environment and construct models of the environment which can be used for planning.

- **control** techniques to apply appropriate torques to joints and motors to move robot vehicles or manipulators to the desired position.
- **path planning** systems which in turn could incorporate neural networks, evolutionary approaches or other emergent intelligence techniques.
- **obstacle avoidance** to prevent collisions with stationary or moving obstacles in the environment.
- **task planning** systems to plan how a job will be broken down into units so as to complete it in a timely manner.

The combination of all of these fields is an enormous task, and one which is likely to be achievable only by a team of experts, rather than a lone researcher. It is for this reason that this dissertation has been limited to path planning alone. The motion planning component in itself is complex, and this research has not produced a plug-in solution, but rather offers techniques which can form part of a motion planning system which in turn can form a part of a complete robot planning and control system.

As this chapter has discussed issues related to the practical application of the GA APF planner, it is appropriate finally to consider a framework within which it might operate. Previously it was shown, in Figure 5.2, how the components of the experimental planner were related. In Figure 6.2, the GA APF planner is shown in a guise which might be suitable for use in a practical planner. This figure by no means sets in stone how such a practical planner should be implemented, but rather offers ideas as to what components might be present, how they could be interrelated and how the GA APF components fit in with external components which may be present in a complete robot control system.

The diagram shows a grey region which is intended to represent the path planning sub-system. The components outside this region are those which

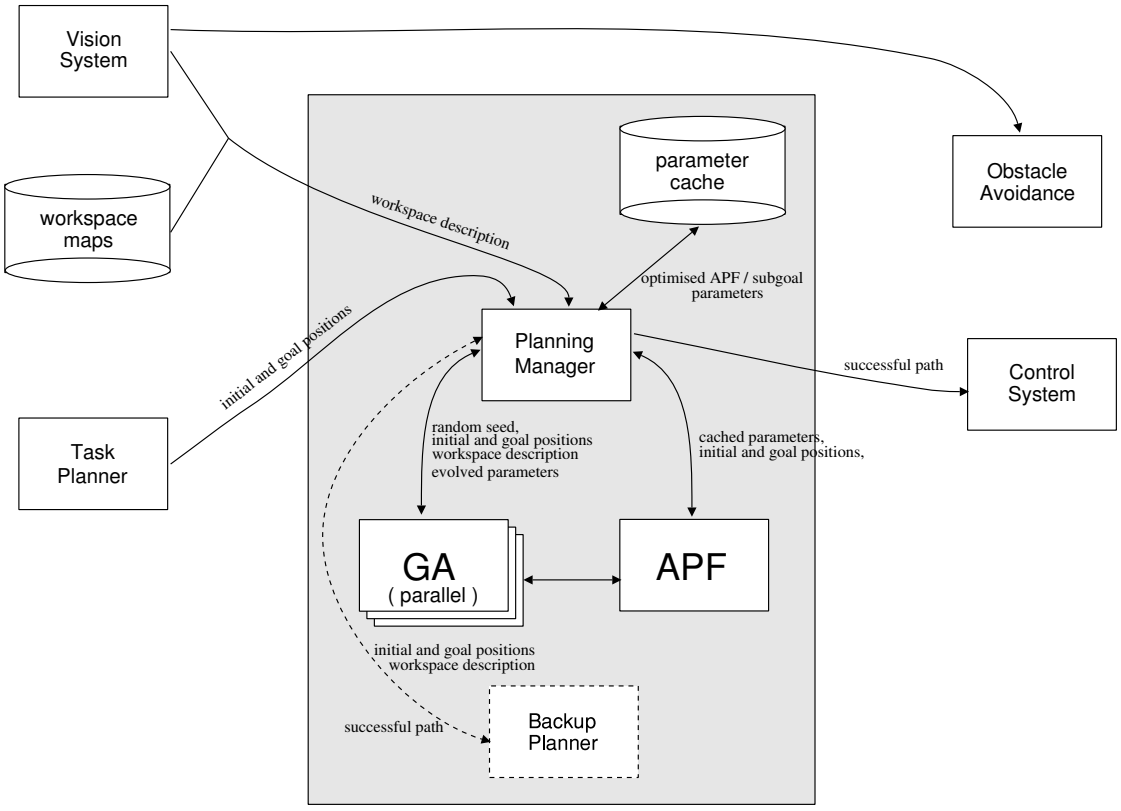


Figure 6.2: An example framework for the GA APF planner

could exist in a robot control system providing information to, and which are provided information from the path planner.

It is assumed that the robot for which this controller is intended would be general in its application. It would not require precise commands to control its every move, but rather would be fed with more general, task level instructions, such as 'move box A from position 1 to position 3', or 'insert bolt type A in hole B'. This information would be provided to the **Task Planner** which would break the overall job into sub-tasks, each of which will in turn involve a movement of the robot. The task planner could, therefore, provide input to the path planner to request that a path be planned between an initial and goal position.

If the robot were to be working in a known environment, such as in a warehouse, or in a production plant, then it is assumed that there would be a selection of **workspace maps** or descriptions which would be stored, and provided to the planner as necessary. If working in an unknown environment, then it would be necessary to generate a map through robot vision, or via robot exploration with sensors on the robot. In either case, by means of the task planner and the vision system or map store, the planner has available to it the information which it requires to proceed with finding a suitable path.

Before discussing the internal components of the planning sub-system, the remaining two external components will be briefly described. Despite the best efforts of a planner, it is likely that some kind of **Obstacle Avoidance** system would be present. Whether the path planner fails, performs an error in planning due to problems with resolution, or whether an unexpected obstacle has appeared which was not considered by the planner, there should be some means by which to halt the robot to prevent a collision. This would likely be achieved by continually monitoring the immediate surroundings of the robot, perhaps via a vision system, or alternatively using sonar or proximity sensors. Once a path has been generated by the planner, this must be followed

by the robot. A method will be needed to provide the necessary signals to motors, thrusters, actuators or whatever means the robot uses to move. The field of control is very large topic of research in itself [Dracopoulos, 1997b; White and Sofge, 1992]. An alternative to having a separate control sub-system may be to construct a potential field which has a relationship with the dynamics of the system, enabling the forces or torques applied to motors to be based directly on the field gradient [Guldner and Utkin, 1995].

Moving inside the grey, planner region of the diagram, the central component is the **Planning Manager**. It is envisaged that this would work in a supervisory role, coordinating the other components. It is therefore the Planning Manager which receives the initial request for a path to be planned, and ultimately outputs a path. The **GA** component of the system is where the evolution takes place. This component would be fed details of the problem for which parameters are to be evolved, and attempt to evolve these parameters. In doing so, it would have to invoke the **APF** planner during fitness evaluation to measure the fitness of candidate solutions. As the cost of hardware is continually reducing, it is suggested that a parallel approach is adopted. The planning manager could run a number of wholly decoupled instances of the GA process, each with a different seed, and could then use the best set of parameters evolved. Alternatively, the parallelisation could be used to speed up a single run, by distributing a single population over a number of processors.

It is possible in some applications, that a certain configuration will act as the goal on repeated occasions. It does not seem very sensible in this case to perform unnecessary computation to generate new parameters each time. The **parameter cache** is an idea proposed to address this issue. The Planning Manager can place parameters for specific problems into the parameter cache, and when it receives a request for a path to a goal which has been previously planned it can retrieve them and use them immediately without consulting the GA component. Memory may be limited in this cache, so a replacement scheme may be used to decide which parameters in the cache

should be replaced by those freshly evolved. When the system is not busy performing on-line planning, the Planning Manager could use this idle time to try to improve on the parameters in the cache off-line. In a similar way, the Task Planner outside the system may perform a similar speculative execution, instructing the Planning Manager to establish parameters for problems which are likely to arise in the future, which can be stored in the cache for rapid retrieval later.

Because the GA is only probabilistically complete, although it is very likely to generate results, it is not guaranteed to. It may be that several runs of the GA are required before satisfactory parameters are evolved and the duration of these runs cannot be predicted, except insofar as a maximum limit may be imposed on them. In a real-time system, failure is not an option, so one or more **Backup Planners**, adopting a different, perhaps more exhaustive path planning approach, could be included in the system. The Backup Planner, running in parallel with the GA APF planner, finds a solution before the GA APF planner, then this alternative path can be used. The problem can continue to be tackled by the GA APF planner off-line, and parameters, if evolved can be stored.

6.7 Summary

Previous chapters have described ways in which evolutionary computing, and specifically GA and GP can be used for robot path planning. Whilst the research for this dissertation has relied on virtual workspaces and simulations, it must be borne in mind that the ultimate goal of research into all areas of robotics is to make robots more independent in all aspects of their operation. This chapter has looked at how the approaches developed during this research might fit into a more complete robot control system, and has suggested some improvements which might be required for the eventual, successful implementation of an Evolutionary Path Planner.

7

Conclusions

7.1 Introduction

This concluding chapter provides an overview of the dissertation, from the initial review of the fields of path planning and evolutionary computing through to the application and evaluation of path planning using GA and GP.

7.2 Summary

This dissertation began, in Chapter 1, by describing why it was necessary for path planning to be researched further given that there were already existing automated path planning techniques in place.

It was suggested that part of the problem with existing, classical planning techniques was their use of the traditional method of problem solving in computer science which imposes a heavy requirement to use knowledge from the problem domain in solving the problem at hand.

As computers are applied to increasing complex problems, path planning included, it is not always possible for human programmers to have a sufficient understanding of the problem. Techniques which use Evolutionary Computing were offered as a means by which to address this problem, as they are

able to learn the knowledge required to solve the problem as an inherent part of their process. It was, therefore, proposed that GA and GP could be useful in path planning, an assertion which this dissertation has sought to test.

Classical approaches to path planning were examined in Chapter 2 with a view to identifying problems with these techniques, and to glean information which would be useful in the subsequent design and implementation of evolutionary path planning algorithms. Indeed a classical approach was later adopted as a central aspect of this research.

The investigation did not seek to demean all existing approaches, but tried to recognise common failings which, in the long term, need to be addressed in future path planners. The speed at which a technique can produce plans is clearly important for its application in a real-world robot. Some planning techniques have quite heavy computational requirements. The dimensionality, magnitude and resolution of the workspace affects some planning methods. As these parameters increase, the storage resources, and possibly computational requirements increase. The inability of many existing planners to generalise to unseen situations was also noted.

Having reviewed more traditional approaches in Chapter 2, attention was drawn, in Chapter 3, to Evolutionary Algorithms which it was hoped would be able to either assist in planning, or even directly perform planning. The related techniques of GA and GP were discussed, with a view to using them for path planning. As speed had already been noted to be important, the ability to speed up GA and GP by means of parallel computing was addressed. To demonstrate that GA and GP showed some promise in their future application to path planning, some existing applications were examined, including planning applications.

Chapter 4 described the use of GP to evolve a generalised planner. GP can be rather slow due to its high computational requirements for difficult problems, but can be used to evolve programs or rules which themselves can be

executed quickly online. A number of different approaches were tested, the best of which was the use of a GP to evolve a function which given a current workspace position and goal would output the next step which the robot should follow to move optimally towards the goal.

Although none of the approaches tried in Chapter 4 were able to evolve planners which could realistically be used in a successful robot, the results certainly did not mark an end to the application of evolutionary algorithms to path planning. An important question addressed by Chapter 5 was "Can evolutionary algorithms improve an existing planning technique?" To this end, it was shown how GA could be used to address a shortcoming in a classical approach — Artificial Potential Field guided path planning. A method was described for using GA to optimise the parameters controlling the construction of the potential field. Comparisons were made between manual and automatic tuning demonstrating the automated version to be far superior. It was then shown how the technique could be further developed to allow the GA to evolve ordered subgoals in the workspace as well as optimised parameters to solve more complex planning problems.

Chapter 6 evaluated the techniques developed over the course of the research, and discussed the implications of implementing evolutionary based planners in the future. The subject of generalisation was revisited, explaining to what extent it had been realised. The chapter looked to the future when it is hoped that an evolutionary path planner might form part of the control system for a real-world robot which would be subject to real-time constraints. Apart from improvements in the performance of hardware, other means for addressing the high computational requirements of GA/GP path planning techniques was described. A framework was offered as an example of how an evolutionary path planner might fit in with other components of a robot control system.

7.3 Contributions

This section summarises the contributions which the research described in this dissertation has made. It first notes specific techniques and applications which make contributions to the fields of evolutionary computing and robot path planning. However, the completion of this study has also given rise to more general observations which also offer a contribution to future researchers.

7.3.1 Evolutionary Computing

simple parallelisation of GP

This parallel implementation of GP demonstrated how significant speedups could be achieved in run times, even when using moderately specified workstations connected together by a relatively slow ethernet (Section 3.9).

application of GP to oral cancer diagnosis

GP was applied to the task of diagnosing oral cancer or pre-cancer from a set of data about patient's lifestyles. This provided an example of GP solving a difficult, real world problem (Section 3.9).

multi-objective fitness scaled through the run

This contribution addresses the problems encountered when the fitness of an individual in a GA or GP population consists of a number of factors which may conflict. The multiple fitness criteria were divided into one primary and one or more secondary criteria. The influence placed on the secondary criteria is a function of the generation number of the run. This encourages the primary goal to be met first, and the secondary goals to be met as the run progresses (Section 4.3.2).

multi-objective adaptive weighting

This is another technique to cope with the problems of multi-objective fitness functions, and was shown to be more effective than the scaling by generation. Again criteria were divided into primary and secondary criteria, but with this technique the relative importance does not monotonically shift from primary to secondary, as with scaling by generation, but varies dynamically according to the performance of solutions against primary objectives 5.9.

7.3.2 Path Planning

evolving a path planning rule using GP

A number of novel approaches were proposed to use GP to evolve a path planning rule off-line, which could later be used to generate plans very quickly off-line (Chapter 4).

optimisation of artificial potential fields using GA

GA was used to automatically set the controlling parameters of the artificial potential fields used for path planning. There is no other known way of automatically setting these parameters (Chapter 5).

the use of sub-goals in an optimised APF planner

A simple planner was demonstrated which, by use of a GA optimised APF and subgoals, was able to solve problems which were not solvable without the use of goals.

7.3.3 General Observations

a non-competitive approach to the application of GP and GA

In Section 3.8 it was noted that GA and GP are very similar, and that for the purpose of this dissertation they would be treated as one. Although there is not necessarily a reference to reinforce this, the impression is sometimes given at conferences and in various publications that GA and GP are in competition. The stance that has been taken during this research is that they are both driven by the same underlying evolutionary principles, and that the primary difference is in the data structure, whether it be a fixed length string, a dynamic tree, or some other hybrid appropriate to the problem being tackled. The experimentation carried out during this research used both GP and GA where it was felt that it was appropriate to do so. It is hoped that the future of GA, GP will include a common sense approach whereby the techniques are applied appropriately, and not in competition, whether applied to path planning or other problems.

Dealing with path planning more specifically, both GA and GP do have something to offer. Previous researchers have shown how the techniques can be used in various guises in path planning, and this research has demonstrated some new approaches, reinforcing the fact that GA and GP are applicable to path planning. The developments made in this research lay the foundation for future research in the application of GA and GP to path planning.

appropriate use of GA and GP for path planning

This dissertation has shown that, at present, given the current state of hardware, the most appropriate way in which GA and GP can be applied to path planning is in a support role, enhancing existing path planning algorithms.

7.4 Recommendations for Future Research

This section makes some specific recommendations as to what research might be carried out in the future to further improve robot path planning in order to eventually realise more independent robots.

parallel path planning

Research should be undertaken into the parallel path planning. Whilst it is clear that parallelisation will undoubtedly lead to higher speed planners, research is still required to find the best architecture to use for GA or GP path planning. Issues of cost against performance must be considered as the goal of this research must eventually be to make realistic real-time planners for real-world robots.

using GP to make functions for potential field

Combined with the use of parallel computing to address the computation requirements, the use of GP to represent the repulsive function for each obstacle, as described in Section 6.3.2, may result in potential fields from which better plans can be derived.

better ways of seeing environment

The work on evolving planners using GP in Chapter 4 described how the programming language used to represent the planner was not able to 'see' the workspace effectively. Research into methods for allowing the workspace description to be effectively 'input' into an evolved program may radically improve the ability of GP to evolve a planner.

separating subgoal positioning and field optimisation

The hybridisation of GA and the APF planning technique led to the use of subgoals to improve the ability of the planner to cope with more difficult problems. To improve the possibility of generalisation in an APF planner, research could be undertaken which separates the optimisation of the potential field and the placing of subgoals into two separate processes. It may be that the optimisation of the potential field can be driven by a fitness measure based on discouraging 'bad' features in the field, such as local minima, saddle points and flat regions, rather than focusing too heavily on optimising the field to get from **a** to **b**. Having generated a 'desirable' field, GA can then be used to generate a coarse grained path by placing subgoals on the workspace, and generating the fine grained plan using the APF to move between the subgoals.

build into a full planner

In Section 6.6, an outline for an APF planner integrated with other robot system components was given. There is a huge amount of research to be explored in this area, incorporating components from classical robotics, from this dissertation, and perhaps from other branches of machine learning such as the neural network field.

extend technique to higher dof robots

It would be interesting to apply future versions of the APF planner to robots with higher degrees of freedom and in higher dimensional workspaces, as will be required in a useful independent robot.

7.5 Concluding Remarks

This dissertation has enquired into previous classical approaches to path planning, and has demonstrated a number of different applications of GA and GP

to path planning problems. In addressing the underlying thesis laid out in Chapter 1:

> Do Genetic Algorithms and Genetic Programming have a role to play in the future development of robot path planning?

The results of the research carried out provide positive evidence that these evolutionary techniques do have a role to play in path planning. Furthermore, in seeking to contribute to the ultimate goal of a fully independent robotic system, this dissertation hopefully offers the motivation to others to embark on further research into the application of Evolutionary Approaches to Robot Path Planning.

Bibliography

AHLUWALIA, M. AND FOGARTY, T. C., [1996]. "Co-evolving hierarchical programs using genetic programming." In Koza, J. R., Goldberg, D. E., Fogel, D. B. and Riolo, R. L. (eds.), *Genetic Programming 1996: Proceedings of the First Conference*, p. 419, Cambridge, MA, USA. MIT Press.

AHUACTZIN, J. M., TALBI, E.-G., BESSIERE, P. AND MAZER, E., [1992]. "Using genetic algorithms for robot motion planning." In Neuman, B. (ed.), *Proceedings of the 10th European Conference on Artificial Intelligence*, pp. 671–675, Vienna, Austria. John Wiley and Sons.

ALANDER, J. T., [1996]. *An Indexed Bibliography of Genetic Algorithms in Robotics*. Tech. Rep. 94-1-ROBOT, Department of Information Technology and Production Economics, University of Vaasa, PO Box 700, SF-65101, Vaasa, Finland.
URL: *ftp://ftp.uwasa.fi/cs/report94-1/gaROBOTbib.ps.Z*

ANDRE, D., [1994]. "Evolution of mapmaking ability: Strategies for the evolution of learning, planning, and memory using genetic programming." In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, pp. 250–255, Orlando, Florida, USA. IEEE Press.

ANDRE, D. AND KOZA, J. R., [1996]. "Parallel genetic programming: A scalable implementation using the transputer network architecture." In Angeline, P. J. and Kinnear, Jr., K. E. (eds.), *Advances in Genetic Programming 2*, chap. 16, pp. 317–338. MIT Press, Cambridge, MA, USA.

ANGELINE, P. J., [1994]. "Genetic programming and emergent intelligence." In [Kinnear Jr., 1994], chap. 4, pp. 75–97.

AURENHAMMER, F., [1991]. "Voronoi diagrams — a survey of fundamental geometric data structures." *ACM Computing Surveys*, **23**(3), 345–405.

BANZHAF, W., FRANCONE, F. D., KELLER, R. E. AND NORDIN, P., [1998]. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, CA, USA.

BARRAQUAND, J. AND LATOMBE, J.-C., [1991]. "Robot motion planning: A distributed representation algorithm." *The International Journal of Robotics Research*, **10**(6), 628–649.

BARTO, A. G., SUTTON, R. S. AND ANDERSON, C. W., [1983]. "Neuronlike adaptive elements that can solve difficult learning control problems." *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-13**(5), 834–846.

BENNETT III, F. H., KOZA, J. R., ANDRE, D. AND KEANE, M. A., [1997]. "Evolution of a 60 decibel Op-Amp using genetic programming." In *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*, Lecture Notes in Computer Science, pp. 455–469, Berlin, Germany. Springer-Verlag.
URL: *http://www-cs-faculty.stanford.edu/~koza/ICES60dB.ps*

BENTLEY, P. AND WAKEFIELD, J., [1998]. "Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms." *submitted to Soft Computing*.
URL: *http://www.cs.ucl.ac.uk/staff/P.Bentley/PetersPapers.html*

BESSIÈRE, P., JUAN-MANUEL, A., TALBI, E.-G. AND EMMANUEL, M., [1993]. "The "Ariadne's Clew" algorithm: Global planning with local methods." In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Yokohama, Japan*, pp. 1373–1380. IEEE, IEEE Press.

BLICKLE, T. AND THIELE, L., [1995]. *A Comparison of Selection Schemes used in Genetic Algorithms*. Tech. Rep. 11, Computer Engineering and

Communications Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastraße 3, 8092 Zurich, Switzerland.

BROOKS, R., [1983]. "Solving the find-path problem by good representation of free space." *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-13**(3), 190–197.

BRUNN, P., [1996]. "Robot collision avoidance." *Industrial Robot*, **23**(1), 27–33.

CAMERON, S., [1994]. "Obstacle avoidance and path planning." *Industrial Robot*, **21**(5), 9–14.

CANNY, J., [1987]. "A new algebraic method for robot motion planning and real geometry." In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (Los Angeles, Oct. 22-24)*, pp. 39–48, Washington D.C., USA. IEEE.

CANNY, J., [1988]. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, USA.

CANTÚ-PAZ, E., [1995]. *A Summary of Research on Parallel Genetic Algorithms*. Tech. Rep. 95007, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, IL 61801.

CHAPMAN, D., [1987]. "Planning for conjuctive goals." *Artificial Intelligence*, **32**, 333–337.

DARWIN, C., [1859]. *On the Origin of Species*. John Murray, London.

DAVIDOR, Y., [1989]. *Genetic Algorithms for order dependent processes applied to robot path-planning*. Ph.D. thesis, Imperial College, London.

DAVIDOR, Y., [1991]. "A genetic algorithm applied to robot trajectory generation." In Davis, L. (ed.), *Handbook of Genetic Algorithms*, chap. 12, pp. 144–165. Van Nostrand Reinhold.

DE GARIS, H., [1994]. "CAM-BRAIN the genetic programming of an artificial brain which grows/evolves at electronic speeds in a cellular automata machine." In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, pp. 337–339, Orlando, Florida, USA. IEEE Press.

DENAVIT, J. AND HARTENBERG, R. S., [1955]. "A kinematic notation for lower-pair mechanisms based on matrices." *Journal of Applied Mechanics*, **77**, 215–221.

DIJKSTRA, E. W., [1959]. "A note on two problems in connexion with graphs." *Numerische Mathematik*, **1**, 269–271.

DRACOPOULOS, D., [1998]. "Neural robot path planning: The maze problem." *Neural Computing and Applications*, **7**(2), 115–120.

DRACOPOULOS, D. AND JONES, A., [1997]. "Adaptive neuro-genetic control of chaos applied to the attitude control problem." *Neural Computing and Applications*, **6**(2), 102–115.

DRACOPOULOS, D. C., [1997a]. "Evolutionary control of a satellite." In Koza, J. R., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H. and Riolo, R. (eds.), *Genetic Programming 1997: Proceedings of the Second Conference*, pp. 77–81, San Francisco, CA, USA. Morgan Kaufmann.

DRACOPOULOS, D. C., [1997b]. *Evolutionary Learning Algorithms for Neural Adaptive Control*. Springer Verlag, London.

DREYFUS, S. E., [1969]. "An appraisal of some shortest-graph algorithms." *Operations Research*, **17**, 395–412.

ELLIOT, C., [1996]. *The use of Inductive Logic Programming and Data Mining techniques to identify people at risk of oral cancer and precancer*. Master's thesis, Brunel University, Department of Electrical Engineering.

ELLIOT, C., KENT, S., HAMMOND, P., DRACOPOULOS, D., DOWNER, M. AND SPEIGHT, P., [1997]. "A comparison of artificial intelligence techniques for the identification of people at high risk of oral cancer." In *Proceedings of the 44th Annual General Meeting of the British Society for Dental Research*, p. 275, Brighton, UK.

ELLIOTT, C., KENT, S., HAMMOND, P., DRACOPOULOS, D., DOWNER, M. AND SPEIGHT, P., [1997]. "A comparison of artificial intelligence techniques for the identification of people at high risk of oral cancer." *Journal of Dental Research*, **76**(5), 1053.

EVEN, S., [1979]. *Graph Algorithms*. Pitman, London, UK.

FIKES, R. E. AND NILSSON, N. J., [1971]. "STRIPS : A new approach to the application of theorem proving to problem solving." *Artificial Intelligence*, **2**, 189–208.

FOGEL, D., [1995]. *Evolutionary Computation*. IEEE Press, New York, NY, USA.

FOGEL, D. B., [1997]. "Evolutionary computation: A New Transactions." *IEEE Transactions on Evolutionary Computation*, **1**(1), 1–2.

FOGEL, L., OWENS, A. AND WALSH, M., [1966]. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, NY, USA.

FORREST, S. (ed.), [1993]. *Proceedings of the 5th International Conference on Genetic Algorithms*, San Francisco, CA, USA. Morgan Kaufman.

FRIEDBERG, R., [1958]. "A learning machine, part i." *IBM Journal of Research and Development*, **2**, 2–13.

FRÖLICH, M. AND WERNER, M., [1996]. *daVinci V2.0.x Online Documentation*. Universität Bremen.
URL: *http://www.tzi.de/~davinci/doc_V2.0/*

FU, K. S., GONZALEZ, R. C. AND LEE, C. S. G., [1987]. *Robotics: Control, Sensing, Vision and Intelligence*. McGraw-Hill.

GLASIUS, R., KOMODA, A. AND GIELEN, S. C. A. M., [1995]. "Neural network dynamics for path planning and obstacle avoidance." *Neural Networks*, **8**(1), 125–133.

GOLDBERG, D. E., [1989]. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, USA.

GONZALEZ, R. C. AND WOODS, R. E., [1993]. *Digital Image Processing*. Addison-Wesley, Reading, MA, USA.

GORDON, V. S. AND WHITLEY, D., [1993]. "Serial and parallel genetic algorithms as function optimizers." In [Forrest, 1993], pp. 177–183.

GULDNER, J. AND UTKIN, V. I., [1995]. "Sliding mode control for gradient tracking and robot navigation using artificial potential fields." *IEEE Transactions on Robotics and Automation*, **11**(2), 247–254.

HAMMOND, P. AND DAVENPORT, J. C., [1997]. "A logic-based approach to prosthesis design." In *Proceedings of the 10th Florida Artificial Intelligence Research Symposium*, pp. 97–100. Florida Artificial Intelligence Research Society.

HAMPO, R. J., [1992]. "The genetic programming paradigm: A new tool for analysis and control." Ford Proprietary.

HANDLEY, S. G., [1993]. "The genetic planner: The automatic generation of plans for a mobile robot via genetic programming." In *Proceedings of the 1993 International Symposium on Intelligent Control*, pp. 190–195. IEEE Press.

HANDLEY, S. G., [1994a]. "The automatic generation of plans for a mobile robot via genetic programming with automatically defined functions." In [Kinnear Jr., 1994], chap. 18, pp. 391–407.

HANDLEY, S. G., [1994b]. "The prediction of the degree of exposure to solvent of amino acid residues via genetic programming." In *Second International Conference on Intelligent Systems for Molecular Biology*, pp. 297–300, Stanford University, Stanford, CA, USA. AAAI Press.

HANDLEY, S. G., [1995]. "Classifying nucleic acid sub-sequences as introns or exons using genetic programming." In Rawlins, C., Clark, D., Altman, R., Hunter, L., Lengauer, T. and Wodak, S. (eds.), *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology (ISMB-95)*, pp. 162–169. AAAI Press.

HARRIS, C. AND BUXTON, B., [1996]. *GP-COM: A Distributed, Component-Based Genetic Programming System in C++*. Research Note RN/96/2, UCL, Gower Street, London, WC1E 6BT, UK.
URL: *ftp://cs.ucl.ac.uk/genetic/papers/gpcom.ps*

HARSTEN, C. T., [1990]. "Application of neural networks to robotics." In Maren, A. J., Harsten, C. T. and Pap, R. M. (eds.), *Handbook of Neural Computing Applications*, chap. 23, pp. 381–389. Academic Press, San Diego, CA, USA.

HAYMAN, A. R., [1997]. *Knowledge work and information technology: a case study in litigation support*. Ph.D. thesis, Brunel University.

HEARN, D. AND BAKER, M. P., [1996]. *Computer Graphics*. Prentice Hall International, Englewood Cliffs, NJ, USA, 2nd edn.

HERMAN, M. AND ALBUS, J. S., [1988]. "Overview of the multiple autonomous underwater vehicle (MUAV) project." In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 618–620.

HIGUCHI, T., IWATA, M. AND WEIXIN, L. (eds.), [1997]. *Evolvable Systems: From Biology to Hardware*. Springer-Verlag, Berlin.

HOCAOĞLU, C. AND SANDERSON, A. C., [1998]. "Multi-dimensional path planning using evolutionary computation." In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pp. 165–170, Anchorage, Alaska, USA. IEEE Press.

HOLLAND, J. H., [1975]. *Adaption in Natural and Artificial Systems*. University of Michigan Press.

HOLLAND, J. H., [1992]. *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 2nd edn.

HOPGOOD, F. AND DUCE, D., [1991]. *A Primer for PHIGS*. John Wiley and Sons, Chichester, England.

HOUILLON, P. AND CARON, A., [1993]. "Planar robot control in cluttered space by artificial neural network." *Journal of Mathematical Modelling and Scientific Computing*, **2**, 498–502.

HWANG, Y. K. AND AHUJA, N., [1992]. "Gross motion planning — a survey." *ACM Computing Surveys*, **24**(3), 219–291.

JOHANNSEN, W., [1911]. "The genotype conception of heredity." *The American Naturalist*, **45**, 129–159.

JONES, A. AND VALENZUELA, C., [1995]. "A parallel implementation of the Evolutionary Divide And Conquer for the TSP." In *proceedings of the 1st IEE/IEEE Conference on Genetic Algorithms in Engineering Systems (GALESIA'95)*, pp. 499–504. IEE.

JUILLE, H. AND POLLACK, J. B., [1995]. "Parallel genetic programming and fine-grained SIMD architecture." In Siegel, E. S. and Koza, J. R. (eds.), *Working Notes for the AAAI Symposium on Genetic Programming*, pp. 31–37, MIT, Cambridge, MA, USA. AAAI.

JUILLÉ, H. AND POLLACK, J. B., [1998]. "Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules." In [Koza et al., 1998], pp. 519–527.

JULLIEN, J., DOWNER, M., ZAKZREWSKA, J. AND SPEIGHT, P., [1995]. "Evaluation of a screening test for the early detection of oral cancer and precancer." *Communications of Dental Health*, **12**, 3.

KENT, S., [1996]. *Diagnosis of Oral Cancer using Genetic Programming*. Tech. Rep. CSTR-96-14 ; CNES-96-04, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK.

KENT, S. AND DRACOPOULOS, D., [1997]. "Genetic programming for prediction and control." *Neural Computing and Applications*, **6**(4), 214–228.

KENT, S. AND DRACOPOULOS, D. C., [1996a]. "Bulk synchronous parallelisation of genetic programming." In Waśniewski, J. (ed.), *Applied parallel computing : industrial strength computation and optimization ; Proceedings of the third International Workshop, PARA '96*, pp. 216–226, Berlin, Germany. Springer Verlag.

KENT, S. AND DRACOPOULOS, D. C., [1996b]. "Speeding up genetic programming: A parallel BSP implementation." In Koza, J. R., Goldberg, D. E., Fogel, D. B. and Riolo, R. L. (eds.), *Genetic Programming 1996: Proceedings of the First Conference*, p. 421, Cambridge, MA, USA. MIT Press.

KHATIB, O., [1980]. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*. Ph.D. thesis, Ecole Nationale Supérieure de L'Aéronautique et de l'Espace, Toulouse. (unpublished, in French).

KHATIB, O., [1986]. "Real-time obstacle avoidance for manipulators and mobile robots." *International Journal of Robotics Research*, **5**, 90–98.

KHATIB, O., [1998]. "Personal communication, April 11th 1998." Confirmation of the fact that the optimisation of gain and distance of influence parameters in artificial potential fields is important, and that work in the area was not forthcoming.

KHOSLA, P. AND VOLPE, R., [1988]. "Superquadric artificial potentials for obstacle avoidance and approach." In *Proceedings of the IEEE International Conference on Robotics and Automation, April 24-29, 1988, Franklin Plaza Hotel, Philadelphia, PA, Volume 3*, pp. 1778–1784.

KINNEAR JR., K. E. (ed.), [1994]. *Advances in Genetic Programming*. MIT press, Cambridge, MA, USA.

KODITSCHEK, D. E., [1987]. "Exact robot navigation by means of potential functions: Some topological considerations." In *Proceedings of the IEEE International Conference on Robotics and Automation, March 31-April 3, 1987, Radisson Hotel and Raleigh Civic Center, Raleigh, North Carolina, Volume 1*, pp. 1–6.

KORKIN, M., DE GARIS, H., GERS, F. AND HEMMI, H., [1997]. ""CBM (CAM-brain machine)": A hardware tool which evolves a neural net module in a fraction of a second and runs a million neuron artificial brain in real time." In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H. and Riolo, R. L. (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA*, pp. 498–503, San Francisco, CA, USA. Morgan Kaufmann.

KOZA, J. R., [1989]. "Hierarchical genetic algorithms operating on populations of computer programs." In Sridharan, N. S. (ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, vol. 1, pp. 768–774, San Mateo, CA, USA. Morgan Kaufman.

KOZA, J. R., [1992]. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

KOZA, J. R., [1994]. *Genetic Programming II*. MIT Press, Cambridge, MA, USA.

KOZA, J. R., BENNETT III, F. H., ANDRE, D. AND KEANE, M. A., [1996]. "Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming." In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 123–131, Cambridge, MA, USA. MIT Press.

KOZA, J. R., BENNETT III, F. H., HUTCHINGS, J. L., BADE, S. L., KEANE, M. A. AND ANDRE, D., [1997]. "Rapidly reconfigurable field-programmable gate arrays for accelerating fitness evaluation in genetic programming." In Koza, J. R. (ed.), *Late Breaking Papers at the 1997 Genetic Programming Conference*, pp. 121–131, Stanford University, CA, USA. Stanford Bookstore.
URL: *http://www-cs-faculty.stanford.edu/~koza/GPfgpa.ps*

KOZA, J. R., BANZHAF, W., CHELLAPILLA, K., DEB, K., DORIGO, M., FOGEL, D. B., GARZON, M. H., GOLDBERG, D. E., IBA, H. AND RIOLO, R. L. (eds.), [1998]. *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison*, San Francisco, CA, USA. Morgan Kaufman.

KOZA, J. R. AND ANDRE, D., [1995]. *Parallel Genetic Programming on a Network of Transputers*. Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science.
URL: *ftp::/elib.stanford.edu/pub/reports/cs/tr/95/1542/*

KOZA, J. R. AND ANDRE, D., [1996]. "Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming." In Angeline, P. J. and Kinnear, Jr., K. E. (eds.), *Advances in Genetic Programming 2*, chap. 8, pp. 155–176. MIT Press, Cambridge, MA, USA.

LANGDON, W., [1996]. "Scheduling maintenance of electrical power transmission networks using genetic programming." In Koza, J. (ed.), *Late Breaking Papers at the GP-96 Conference*, pp. 107–116, Stanford, CA, USA. Stanford Bookstore.

LANGDON, W., [1999]. "The GP bibliography."
URL: *http://www.cs.bham.ac.uk/~wbl/biblio/gp-bibliography.html*

LATOMBE, J.-C., [1991]. *Robot Motion Planning*. Kluwer Academic Publishers, Cambridge, MA, USA.

LEE, W.-P., HALLAM, J. AND LUND, H. H., [1997]. "Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots." In *Proceedings of IEEE 4th International Conference on Evolutionary Computation, April 1997, Indianapolis*, vol. 1, pp. 510–506, New York, NY, USA. IEEE Press.
URL: *ftp://ftp.daimi.aau.dk/pub/stud/hhl/ebbicec97.ps.Z*

LI, T.-Y., BARRAQUAND, J. AND LATOMBE, J.-C., [1990]. *3D Randomized Path Planner*. Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305.

LIU, W., MURAKAWA, M. AND HIGUCHI, T., [1997]. "Evolvable hardware for on-line adaptive traffic control in ATM networks." In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H. and Riolo, R. L. (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 504–509, Stanford University, CA, USA. Morgan Kaufmann.

LOZANO-PÉREZ, T., [1983]. "Spatial planning: A configuration space approach." *IEEE Transactions on Computers*, **C-32**(2), 108–120.

LOZANO-PÉREZ, T., [1987]. "A simple motion-planning algorithm for general robot manipulators." *IEEE Journal of Robotics and Automation*, **RA-3**(3), 224–238.

MANN, R. C., HAMEL, W. R. AND WEISBIN, C. R., [1988]. "The development of an intelligent nuclear maintenance robot." In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 621–623.

MAZER, E., AHUACTZIN, J.-M. AND BESSIÈRE, P., [1998]. "The Ariadne's Clew algorithm." *Journal of Artificial Intelligence Research*, **9**, 295–316.

MCDERMOTT, J., [1982]. "R1: A rule-based configurer of computer systems." *Artificial Intelligence*, **19**(1).

MCLEAN, A. W. AND CAMERON, S. A., [1996]. "The virtual springs method: Path planning and obstacle avoidance for redundant manipulators." *International Journal of Robotics Research*, **15**(4), 300.

MILLER, R. AND REED, J., [1993]. *The Oxford BSP Library Users' Guide*. Tech. rep., University of Oxford.

MINSKY, M. AND PAPERT, S., [1969]. *Perceptrons*. MIT Press, Boston, MA, USA.

MONTANA, D. J., [1994]. *Strongly Typed Genetic Programming*. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA.
URL: *ftp://ftp.io.com/pub/genetic-programming/papers/stgp2.ps.Z*

NAWA, N. E., DE GARIS, H., GERS, F. AND KORKIN, M., [1998]. "ATR's CAM-brain machine (CBM) simulation results and representation issues." In [Koza et al., 1998], pp. 875–882.

NEWELL, A. AND SIMON, H., [1963]. "GPS — a program that simulates human thought." In Feigenbaum, E. A. and Feldman, J. (eds.), *Computers and Thought*, pp. 279–296. McGraw-Hill, New York, USA.

NILSSON, N. J., [1994]. "Teleo-Reactive programs for agent control." *Journal of Artificial Intelligence Research*, **1**, 139–158.

NORDIN, P., [1994]. "A compiling genetic programming system that directly manipulates the machine code." In [Kinnear Jr., 1994], chap. 14, pp. 311–331.

NOYES, J. L., [1997]. "Acceleration of training." In Fiesler, E. and Beale, R. (eds.), *Handbook of Neural Computation*, chap. B3.4. Institute of Physics Publishing, Bristol, UK.

Ó'DÚNLAING, C., SHARIR, M. AND YAP, C., [1983]. "Retraction: A new approach to motion planning." In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pp. 207–220, Boston, USA.

PANG, X. AND WERBOS, P. J., [1998]. "Neural network design for J function approximation in dynamic programming." *Journal on Mathematical Modeling and Scientific Computing (a Principiea Scientia journal)*, **5**(2). **URL:** *http://xxx.soton.ac.uk/list/adap-org/9806001*

POLI, R. AND LANGDON, W. B., [1997]. *Genetic Programming with One-Point Crossover and Point Mutation*. Tech. Rep. CSRP-97-13, University of Birmingham, School of Computer Science.

QUARLES, T., NEWTON, A., PEDERSON, D. AND SANGIOVANNI-VINCENTELLI, A., [1994]. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA.

RECHENBERG, I., [1973]. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.

REES, D., [1989]. *Essential Statistics*. Chapman and Hall, London, 2nd edn.

REULEAUX, F., [1876]. *The Kinematics of Machinery: Outlines of a Theory of Machines*. Macmillan and Company, London.

REYNOLDS, C. W., [1994]. "Evolution of obstacle avoidance behavior: using noise to promote robust solutions." In [Kinnear Jr., 1994], pp. 221–241.

RICH, E., [1983]. *Artificial Intelligence*. Computer Science Series. McGraw-Hill, Singapore, international edn.

RIZKI, M. M. AND TAMBURINO, L. A., [1998]. "Evolutionary computing applied to pattern recognition." In [Koza et al., 1998], pp. 777–785.

ROSIN, P. AND WEST, G. A. W., [1995]. "Salience distance transforms." *Graphical Models and Image Processing*, **57**(6), 483–521.

RYLATT, R., CZARNECKI, C. AND ROUTEN, T., [1995]. "Adaptive genetic algorithms for multi-point path finding in artificial potential fields." In Pearson, D., Steele, N. and Albrecht, R. (eds.), *International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 128–131, Wien. Springer-Verlag.

SCHAFFER, J. D. (ed.), [1989]. *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Francisco, CA, USA. Morgan Kaufman.

SCHWEFEL, H.-P., [1975]. "Evolutionsstrategie und numerische optimierung." Dissertation, Technische Universität Berlin, Germany.

SELIG, J., [1992]. *Introductory Robotics*. Prentice Hall, Hemel Hempstead, UK.

SHEU, P. C.-Y. AND XUE, Q., [1993]. *Intelligent Robotic Planning Systems*, vol. 3 of *World Scientific Series in Robotics and Automated Systems*. World Scientific Publishing, Singapore.

SHONKWILER, R., [1993]. "Parallel genetic algorithms." In [Forrest, 1993], pp. 199–205.

SHORTLIFFE, E. H., DAVIS, R., AXLINE, S., BUCHANAN, B., MERIGAN, T. AND COHEN, S., [1973]. "An artificial intelligence program to advise

physicians regarding antimicrobial therapy." *Computers and Biomedical Research*, **6**, 544–560.

STALLINGS, W., [1992]. *Operating Systems*. Macmillan, New York, USA.

STENDER, J., [1993]. *Parallel Genetic Algorithms: Theory and Applications*. Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, Holland.

SUH, S.-H. AND KANG, G. S., [1988]. "A variational dynamic programming approach to robot-path planning with a distance-safety criterion." *IEEE Transactions on Robotics and Automation*, **4**(3), 334–349.

SUTTON, R. S. AND BARTO, A. G., [1998]. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

SYSWERDA, G., [1989]. "Uniform crossover in genetic algorithm." In [Schaffer, 1989].

TELLER, A., [1994]. "The evolution of mental models." In [Kinnear Jr., 1994], chap. 9, pp. 199–219.

THOMPSON, A., [1996a]. "Silicon evolution." In Koza, J. R., Goldberg, D. E., Fogel, D. B. and Riolo, R. L. (eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 444–452, Stanford University, CA, USA. MIT Press.

THOMPSON, A., [1996b]. "Silicon evolution." In Koza, J. R., Goldberg, D. E., Fogel, D. B. and Riolo, R. L. (eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 444–452, Cambridge, MA, USA. MIT Press.

VALIANT, L. G., [1990]. "A bridging model for parallel computation." *Communications of the Association for Computing Machinery*, **33**(8), 103–111.

VAN DER SMAGT, P., [1994]. "Simderella: a robot simulator for neuro-controller design." *Neurocomputing*, **6**(2), 281–285.

VERONNEAU, P., [1998]. "Development of a graphical user interface." Report written at Brunel University by student visiting from IRESTE, Rue Christian Pauc, 44306 Nante, Cedex 3, France.

WERBOS, P., [1998]. "Personal communication, April 10th 1997." Discussion about unpublished work after his collaboration with Xiaoxhong Pang — in particular how the learning rate was very slow when attempts were made to generalise over 6 mazes rather than only a single maze.

WERBOS, P. J. AND PANG, X., [1996]. "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot." *World Congress on Neural Networks Proceedings (WCNN'96)*, pp. 88–92.

WHITE, D. A. AND SOFGE, DONALD, A. (eds.), [1992]. *Handbook of Intelligent Control:Neural, Fuzzy and Adaptive Approaches*. Multiscience Press, New York, USA.

WHITLEY, D., [1989]. "The genitor algorithm and selection pressure: Why rank based allocation of reproductive trials is best." In [Schaffer, 1989], pp. 116–121.

WINSTON, P. H., [1992]. *Artificial Intelligence*. Addison-Wesley, Reading, MA, USA, 3rd edn.

WINTER, C. S., MCILROY, P. W. A. AND FERNANDES-VILLACANAS, J. L., [1994]. "Evolving software techniques." *BT Technology Journal*, **12**(2), 121–131.

XIAO, J., MICHALEWICZ, Z., ZHANG, L. AND KRZYSZTOF, T., [1997]. "Adaptive evolutionary planner/navigator for mobile robots." *IEEE Transactions on Evolutionary Computing*, **1**(1), 18–28.

ZELINSKY, A., [1992]. "A mobile robot exploration algorithm." *IEEE Trasactions on Robotics and Automation*, **8**(6), 707–717.

ZHAO, K. AND WANG, J., [1998]. "Path planning in computer animation employing Chromosome-Protein Scheme." In [Koza et al., 1998], pp. 439–444.

A

Oral Cancer Diagnosis using GP

A.1 Introduction

This appendix describes work carried in the course of the research towards this PhD. The study provided the opportunity for the practical application of GP to a real world problem, rather than contrived examples sometimes used in the evaluation of GP.

The results of the research carried out was reported in a technical report, conference and in two journal publications [Elliot et al., 1997; Kent, 1996; Kent and Dracopoulos, 1997; Elliott et al., 1997]

A.2 Background

There is a wide range of problems which historically have required experts to examine samples of data, and from their analysis assign a classification or make a prediction. The experts draw on their previous knowledge gained through formal training and experience.

A particularly important area where such prediction is used is in medical screening programmes. The idea of such programmes is to detect potentially harmful conditions in patients sufficiently early, to enable effective treatment.

The manual screening process is labour intensive, and therefore expensive, with the consequence that only a limited number of patients are screened. The less patients that are screened, the less problems will be identified early enough for treatment.

Computer diagnosis enables very large patient samples to be analysed, either as the only method of screening, or as a preliminary screening process to identify those most at risk of a particular condition and therefore in need of more thorough manual screening. A specific example of such screening is for oral cancer and pre-cancer.

Data was made available from the Eastman Dental Institute following a previous study [Jullien et al., 1995] which provided information on patient's lifestyles and habits. It was hoped that GP would be able to learn a rule to classify patients into two groups. The rule should predict whether or not a patient was 'at risk' from oral cancer or pre-cancer.

A.3 Patient Data

The data used for the project was derived from data collected from questionnaires distributed to over 2000 dental patients. The data were presented to the GP diagnosis system by preparing a number of predicates shown in Table A.1. For each patient, the system was therefore presented with a `true` (1) or `false` (0) value for each of the 12 predicates. In addition to the data there was a diagnosis for each of the patients. This diagnosis was from a specialist who had access to all the necessary definitive diagnosis aids, such as biopsy, at his disposal.

Some of the predicates need explanation. Firstly, the age of the patient from the questionnaire was broken down into a ranges of 15 years. All patients in the trial were over 40 years of age, which is why the ranges start so high. There are subtle differences in the smoking predicates. The first specifies

1. Gender is female (i.e. 0=Male, 1=Female)
2. Age $>$ 50
3. Age $>$ 65
4. Age $>$ 80
5. Age $>$ 95
6. Have been smoking within last 10 years
7. Currently smoke $\geq$ 20 cigarettes per day
8. Have been smoking for $\geq$ 20 years
9. Beer and/or wine drinker
10. Fortified wine and/or spirit drinker
11. Drink excess units
    (i.e. Male $>$ 21, Female $>$ 14
12. Not visited dentist in last 12 months

Table A.1: Patient lifestyle and habit predicates

if the patient has smoked at any time in the last ten years, or whether or not they currently smoke. The next predicate is true if the patient currently smokes 20 or more cigarettes per day. The final predicate reflects whether the patient has, at any time during their life, smoked for a period of 20 or more years. The alcohol predicates are true if the patient drinks any of the respective drinks, and the excess units predicate is set if the patient drank over the recommended weekly units of alcohol. At the time of the study this was 21 units for men and 14 for women.

Using this method of representing the data, each patient's details were reduced to a 13-bit string consisting of the 12 predicates plus a diagnosis bit. A training set and a test set were used. The training set contained 991 individuals of which 948 had negative diagnoses and 43 had positive diagnoses. The test set contained 132 records, 121 with negative diagnoses and 11 with positive diagnoses. A lot of records seem to have been lost from the original 2000 patients questioned. This is because the data contained a lot of negative records, and very few positive records. A large number of negative diagnosis patients were removed to balance the sets somewhat.

A.4 GP Approach

It has become fairly standard practice in the GP field to present a "tableau" which defines the parameters for a particular application of the technique. This tradition is observed for this problem in Table A.2. This is expanded below.

Objective:	Evolve a rule to predict the presence of oral cancer or pre-cancer from patient data.
Terminal set:	12 boolean values corresponding to data predicates
Function set:	`AND, OR, NOT`
Fitness:	Special 'shifted' fitness as described in text
Fitness Case:	991, 13-bit strings corresponding to patient data.
Parameters:	population = 500, generations = 600

Table A.2: Tableau for Oral Cancer Diagnoses

Function and Terminal Sets

The function set is a very straightforward set of of three logical operators: `AND`, `OR`, and `NOT`. A solution to the problem will therefore be a logical expression using patient attributes as variables. The patient attributes are instantiated in the evolved rules in the terminals. There are therefore 12 bit-fields corresponding to the predicates in Table A.1. The programs which are evolved evaluate to either `true` for a positive diagnosis, or `false` for a negative diagnosis.

Fitness Measure

The fitness measure needs to reflect how well an evolved program performs over all the records in the training set. To this end, there are two simple candidates for measuring fitness. The first is the mean-square error:

$$\frac{1}{n}\sum_{i=1}^{n}(o_i - p_i)^2 ,$$

where n represents the number of patient records against which the individual was evaluated, o_i is the observed diagnosis of the patient, and p_i is the diagnosis predicted by the genetic program.

The problem with such a measure is that it steers the evolutionary process in a subtly wrong direction. There are some programs in the population which are contradictory — they always evaluate to `false` irrespective of the input values. Contradictory individuals will inevitably appear in the first, randomly created, generation of the evolutionary process. Because there are a disproportionately high number of training records with negative diagnoses, these contradictory programs will perform very well, correctly diagnosing most of the training examples. The GP process will rapidly converge; the individuals will have high fitness, but will always predict a negative diagnosis.

A second method is simply a proportion of correct diagnoses:

$$\frac{TP+TN}{TP+FP+TN+FN},$$

where TP, FP, TN, FN represent the number of true positive, false positive, true negative and false negative diagnoses respectively. This measure again falls down due to the high number of negative diagnosis records present in the training data, encouraging rules which only predict negative diagnoses.

To encourage convergence towards a good solution to the problem, a slightly different fitness measure was devised which more accurately rewards a good individual. The ms-error evaluates to 0.0 for a program with a perfect prediction rate, and 1.0 for a completely useless individual. It was instead decided to adopt a measure which rewards for true, positive diagnoses and punishes for false, negative diagnoses. The measure is defined as:

$$\frac{\frac{TP}{TP+FP} - \frac{FN}{TN+FN} + 1.0}{2.0}.$$

This fitness allows the addition of between 0.0 and 1.0 for correctly predicted positive diagnoses, and the subtraction of between 0.0 and 1.0 for incorrectly

predicted negative diagnoses, resulting in a total range of -1.0 to +1.0. This is then normalised so that the contradictory individuals which previously caused problems are centred at 0.5, thus relatively reducing their fitness. The optimum rule which correctly predicts all positive diagnoses and does not falsely predict any negative diagnoses is valued at 1.0. The worst case is at 0.0, where the rule incorrectly diagnoses all patients. This method rewards for correct diagnoses and punishes for incorrect diagnoses more effectively than the previous two methods. This shifted fitness method is the one which was used in all diagnosis experiments.

A.5 Results

The performance of the evolved rule is measured using a range of metrics used in the evaluation of diagnosis tools. These are defined in Table A.3 with results for the best evolved GP rule. For comparison, values are given for a Neural Network rule developed in Elliot [1996], and for a manual dental screener.

Metric	Description	Definition	Performance Ratings		
			GP	NN	Manual
Sensitivity	Performance of a test in terms of its ability to accurately predict a positive outcome, given that the outcome was actually positive.	$\frac{TP}{TP+FN}$	73%	64%	74%
Specificity	Performance of a test in terms of its ability to accurately predict a negative outcome given that the outcome was actually negative	$\frac{TN}{TN+TP}$	65%	68%	99%
Positive Predictive Value	Performance of the test in terms of percentage of positive outcomes	$\frac{TP}{TP+FP}$	15%	15%	67%
Negative Predictive Value	Performance of a test in terms of a percentage of negative outcomes	$\frac{TN}{TN+FN}$	96%	95%	99%

Table A.3: Table performance metrics for best evolved diagnostic rule

These results show that although the evolved rule does not perform as well as a manual screener, it does perform very similarly to a neural network im-

plementation of a diagnosis tool. Although the evolved rule could not act as a substitute for a dental screener, it could act as a suitable pre-filter which could be embedded in patient management software.

B

Configuration Files for GA APF Software

B.1 Introduction

The genetic algorithm, artificial potential field planning software written for this thesis employs, at its heart, a set of files to specify the problem, and in particular the environment containing obstacles. This appendix provides examples of the files and the environments to which they correspond, in order to demonstrate how the representation is very compact when compared to that which might be required for a bitmap representaiton of a similar environment.

B.2 Obstacle Polygon Library

The polygons which are used to represent obstacles in the workspace are defined only one in a library. Although this adds a fixed overhead to storage, it makes creation of an environment simpler, and saves storage in the specification file itself. The library is stored in a C header file, although there is no reason why the library could not be implemented in a file which could be loaded at run time, removing the need for recompilation when a new shape is added. The polygons are stored as ordered lists of coordinate pairs in a normalised coordinate system. The convention used here has been to define the

vertices in an anti-clockwise direction. Examples for a square, right angled triangle and octagon are:

```
#define SHAPE_STORE 3

enum {SQUARE, TRIANGLE, OCTAGON};

/* Define some coordinates for some shapes */
float square_coords[8]={0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0};
float rt_ang_tri_coords[6]={0.0, 0.0, 1.0, 0.0, 0.0, 1.0};
float oct_coords[16]={0.2929,0.0000,
                      0.7071,0.0000,
                      1.0000,0.2929,
                      1.0000,0.7071,
                      0.7071,1.0000,
                      0.2929,1.0000,
                      0.0000,0.7071,
                      0.0000,0.2929};
```

B.3 An example Enviroment

An example environment is depicted in Figure B.1 which is a screen shot of the GUI used in this research [Veronneau, 1998]. It is a typical enviroment containing various shaped obstacles and an initial and a goal position.

B.3.1 Workspace file

The environment is defined by a text file:

```
FILETYPE:1
[xdim]
100
[ydim]
100
[obstacles]
4
[primitive]
0
[xpos]
10.0
```

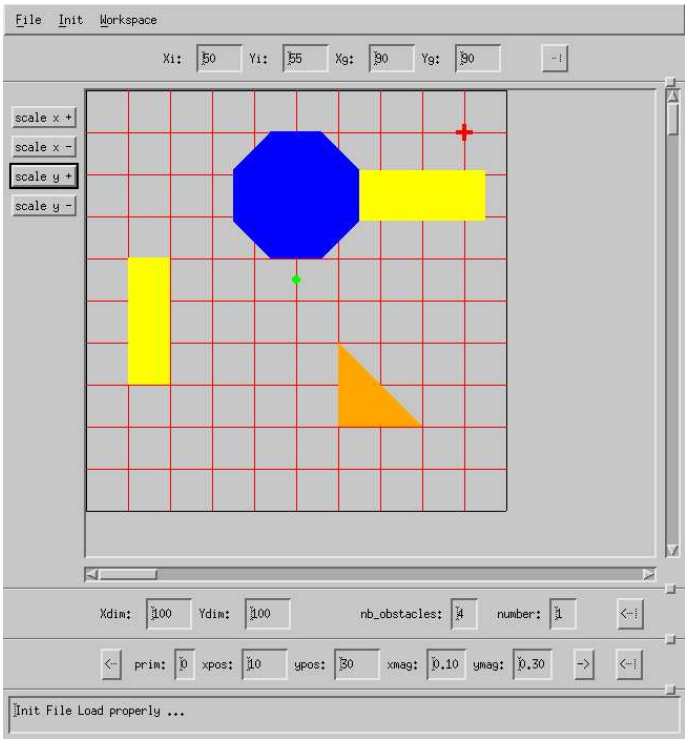


Figure B.1: An Example path planning workspace displayed using the custom built GUI

```
[ypos]
30.0
[xmag]
0.100000
[ymag]
0.300000
[rot]
0.0
[primitive]
2
[xpos]
35.0
[ypos]
60.0
[xmag]
0.300000
[ymag]
0.300000
[rot]
0.0
[primitive]
0
[xpos]
```

```
65.0
[ypos]
69.0
[xmag]
0.300000
[ymag]
0.120000
[rot]
0.0
[primitive]
1
[xpos]
60.0
[ypos]
20.0
[xmag]
0.200000
[ymag]
0.200000
[rot]
0.0
```

B.3.2 Initialisation file

To allow a single workspace file to be re-used with a number of different initial conditions, a separate file was used which contained just four co-ordinate values:

```
FILETYPE:0
[init_x]
50
[init_y]
55
[goal_x]
90
[goal_y]
90
```

B.3.3 Parameter file

Once suitable parameters were evolved by the GA APF planner, these too were output to a file:

There is an issue relating to the use of a file to store these parameters in that accuracy may be lost in converting the internal representation of floating point numbers to string form for output to a file, and subsequently back to an internal representation when these numbers are read back into a program for creation of a path. This approach was adopted here to allow for manual adjustment to the parameters, but it is accepted that an alternative method might be desirable in a real, embedded application.

C

The Artificial Ant Problem

C.1 Introduction

This appendix describes the Artificial Ant problem which is a well known, toy-problem in Genetic Programming and was described by Koza [1992].

C.2 Description

The problem involves moving a robot ant along a trail of food which lies on a grid. This trail contains 157 pieces of food as shown in Figure C.1. In this figure, a black square represents a piece of food, and a grey square represents a gap in the trail. The figure shows the top left corner of the full grid which is 100×100 squares. The tableau for this problem is shown in Table C.1.

Objective:	Evolve a program to guide a robot ant along a trail of 'food'.
Terminal set:	`LEFT, RIGHT, FORWARD`
Function set:	`IF_FOOD_AHEAD, BRANCH2, BRANCH3`
Fitness:	Number of pieces of food collected before a timeout of 3,000 moves is made.
Fitness Case:	Single case — a grid containing a trail of food
Parameters:	population = 2000, generations = 50

Table C.1: Tableau for Robot Ant Problem

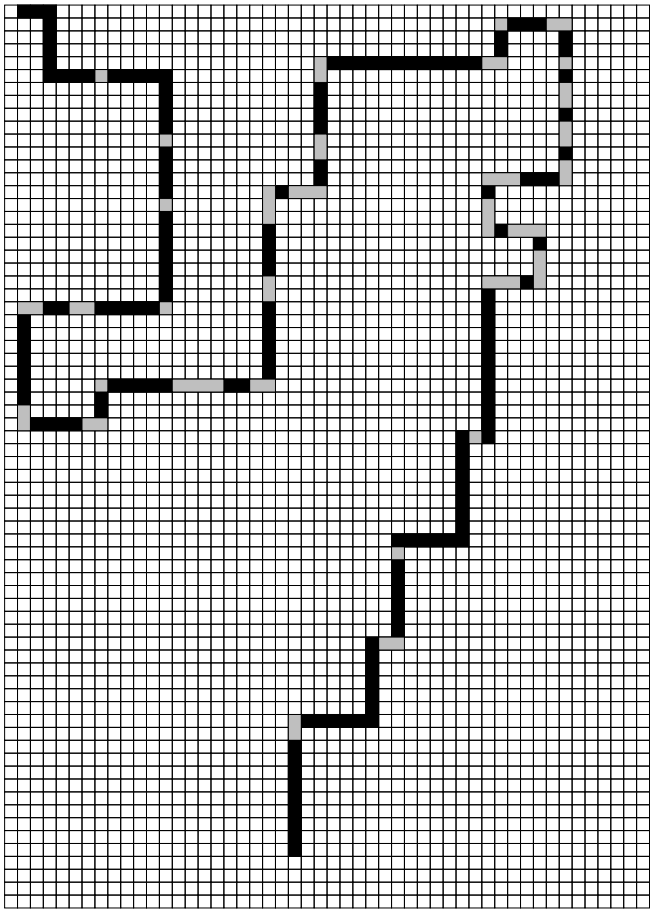


Figure C.1: The Artificial Ant trail.

The ant is capable of a simple set of actions, these being:

- Move forward one grid square
- Turn Left 90^o
- Turn Right 90^o

The ant also has a limited decision making ability through a function:

- If food in square immediately ahead then action a else action b

The function set is augmented by the addition of two unconditional branching functions. This allows sequences of ant actions to appear in the evolved programs. The 3,000 move timeout is to prevent the ant finding all the food on the trail simply by randomly wandering over all 10,000 grid squares.