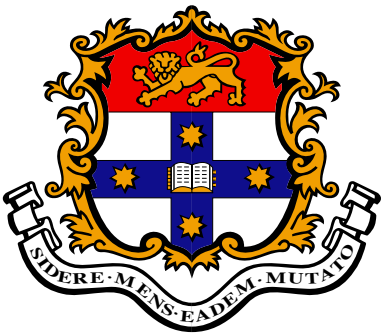
Time-Optimal Active Decision Making

Thomas L. P. Allen

A thesis submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy



ARC Centre of Excellence in Autonomous Systems
Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

April 12, 2011

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Thomas L. P. Allen

April 12, 2011

Abstract

Thomas L. P. Allen
The University of Sydney

Doctor of Philosophy
April 12, 2011

Time-Optimal Active Decision Making

This thesis describes a class of problems known as Time-Optimal Active Decision Making (TOADM). The objective of TOADM problems is to make decisions and act upon them in order to achieve a goal, and to do so in the minimum total time. This total time includes both the time taken to make decisions, and the time taken to act upon them. Such problems exhibit three key features. Firstly, they make decisions and act at the same time; the decisions are 'active' because they are made while the system is operational. Secondly, the performance of the system is influenced by the time taken to compute decisions during its operation. Thirdly, successful operation requires the achievement of a goal in unbounded time, but improved performance can be obtained by doing so in less time.

TOADM problems arise in many disciplines, but the principle example used throughout this thesis is Time-Optimal Planning and Execution (TOPE) for autonomous agents. This problem requires an agent to travel from its initial location to a specified goal, and to do so safely and in as little time as possible. In this case, the total time includes both the time spent computing the plans used to guide the agent and the time spent executing them. Since the state space of the agent can be dynamic, a trade-off must be made between computing efficient plans slowly, and inefficient plans quickly.

This thesis derives a framework for solving general TOADM problems, and demonstrates its application to the TOPE problem. The technique involves estimating the set of system parameters that specify the best trade-off between the time required to make decisions and the time required to act upon them. Since the state space of the decision making process can be dynamic, these estimates must be continually recomputed, and the parameters continually adjusted while the system is in operation. These system parameters are entirely application dependent, but include any run-time adjustable features of the system that affect this trade-off. It is shown that time-optimal behaviour is obtained when the parameter set is selected such that the expected total time is minimised, and that this total time is the sum of the expected time to make the parameter set selection, the expected time to make a decision given this parameter set, and the expected time to achieve a goal given this decision.

Unfortunately, selection of the optimum parameter set is known to be an intractable problem, and is made even more difficult in cases such as the TOPE problem where even the assessment of a single choice of parameter set is itself an intractable problem. Experiments are performed which demonstrate that if this selection can be performed sufficiently fast and accurately, it is possible to improve upon the performance of any technique which uses a fixed set of parameters. This is true even when this parameter set is the optimal fixed choice (determined by a brute-force analysis that is unavailable in most practical applications).

The TOADM framework is implemented for the TOPE problem, where the trade-off is between the time required to compute plans, and the time required to execute them. The available system parameters include the resolution of a discrete representation of the operating environment, and variables affecting the behaviour of the planning algorithms used. The final set of experiments verifies that it is possible to implement two categories of parameter selection techniques that achieve sufficient accuracy and timeliness. The experiments show that several selection techniques are able to outperform all fixed parameter techniques, and even techniques with additional information about the scenario. It is also shown that there exist scenarios where fixed parameter techniques are completely unsuitable. Although the simplest types of selection techniques are also unsuited to these scenarios, others remain able to significantly outperform all alternative techniques and approach the theoretical bounds on performance.

Acknowledgements

Many people have been invaluable in the development of this thesis, and my thanks must start with everyone I've worked with, had discussions with, or enjoyed beers with at the Australian Centre for Field Robotics (ACFR). It is without any doubt that the ACFR is now a world renowned robotics research group, and it has been a pleasure to work with so many bright and enthusiastic people. Particular thanks must go to Hugh for raising it to the heights it has now reached, and to the under-appreciated ladies who keep the entire place running, and who have supported all my endeavours to bypass university bureaucracy wherever possible: Ruth, Christy, Lisa, and Natasha.

To all the past and present members of the Argo & CORD groups: James, Andrew, Chris, Sisir, Thierry, Laura, Richard, Iain, Dave, Craig, Bang, and Mark – thank you for the discussions, help, support, and laughs. It's been a blast working with all of you, and I hope that we manage to continue in the future.

To my supervisor, Steve – thanks for not making me look into infrared cameras. My choice to ignore your suggestion and to procrastinate instead by writing a computer program to play Snakes, led to my interest in reinforcement learning, optimisation, and the application of these concepts to path planning. These in turn led to the development of the main ideas in this thesis, which came full circle when the best performing TOADM technique turned out to employ the same algorithm as my computer controlled snakes. Thanks for providing the support when it was needed, the painful criticism when it was deserved, and the silence when you saw I was on to something interesting.

To my family: Trish, David, Matthew, and Ben, my new family: Emilia and Ben, and my wife Natalie – thanks for sticking with me through this process. To Natalie especially, who has now endured living with me while I've written two theses, I hereby promise never to write another one.

Nothing is more difficult, and therefore more precious, than to be able to decide.

Napoleon Bonaparte

In any moment of decision the best thing you can do is the right thing.
The worst thing you can do is nothing.

Theodore Roosevelt

Contents

List of Acronyms

ACFR	Australian Centre for Field Robotics v
ADM	Active Decision Making 1
ARC	Australian Research Council i
CAS	Centre for Autonomous Systems xii
CORD	CAS Outdoor Research Demonstrator 17
DT	Distance Transform 54
EM	Expectation Maximisation 113
FCS	Feedback Control System 1
GPS	Global Positioning System 13
HR	Hierarchical Replanner 39
LPM	Local Planning Method 27
MDP	Markov Decision Process 11
PerceptOR	Perception for Off-Road Navigation 38
PHR	Parallel Hierarchical Replanner 57
POMDP	Partially Observable Markov Decision Process 11
PRM	Probabilistic Roadmap 28
RRT	Rapidly-Exploring Random Tree 28
SA	Simulated Annealing 114
TDL	Temporal-Difference Learning 113
TOADM	Time-Optimal Active Decision Making 1
TOPE	Time-Optimal Planning and Execution 6
UAV	Unmanned Aerial Vehicle 31
UGV	Unmanned Ground Vehicle 6
UML	Unified Modeling Language 2

List of Figures

List of Notation

General Notation

$(.)^*$	The optimal value of (.)
$\hat{(.)}$	The expected value of (.)
$(.)_i$	The value of (.) in a particular iteration, i

Planning Systems

a	An action
c_E	The cost of executing a plan
F	A reward function used by the Bellman equation
m	The scalar metric value calculated by a metric function
M	The metric function used to assess the performance of a planning system
p	An ordered set of states $x \in X$ comprising a plan to a goal, $x_G \in X_G$
t	A discrete period of time
t_C	The time taken to compute optimal system parameters
t_D	The time taken to make a decision
t_P	The time taken to compute a plan
t_E	The time taken to execute a plan

T	A state transition function from a state given an action
V	A value function used by the Bellman equation
x	A state
X	A state space or set of states
x_G	A goal state of a plan
X_G	A set of goal states for a plan
x_S	The starting state for a plan
y	A set of arguments to a planning system
Y	The space of available sets of arguments to a planning system

Chapter 1

Introduction

This thesis is concerned with *decision making*, which is defined herein as a process used to determine appropriate operating parameters or control inputs in order to influence the behaviour of a system. In particular, this thesis defines a subset of decision making problems in which the time taken to make decisions influences the behaviour of the system. The category of such problems is denoted Active Decision Making (ADM), where 'active' is intended to convey that these decisions are made while the system is operational.

Within the field of ADM this thesis seeks to achieve *time-optimal* performance, meaning that decisions are made such that the time required for the system to complete its objectives is minimised. The word 'optimal' must always be used with respect to some criteria, and in this case 'time-optimal' is used with respect to system performance. The objective of this thesis is thus to derive a time-optimal operating procedure for an ADM, such that no other sequence of decisions is able to further reduce the time required. This final problem is denoted Time-Optimal Active Decision Making (TOADM), and the principal contribution of this thesis is a process by which it can be solved in the general case.

1.1 Motivation

To motivate the need for solutions to the TOADM problem, consider the flow of information or data in a decision making system. Figure 1.1 visualises this data flow for a general Feedback Control System (FCS). The figure illustrates the three main components of any

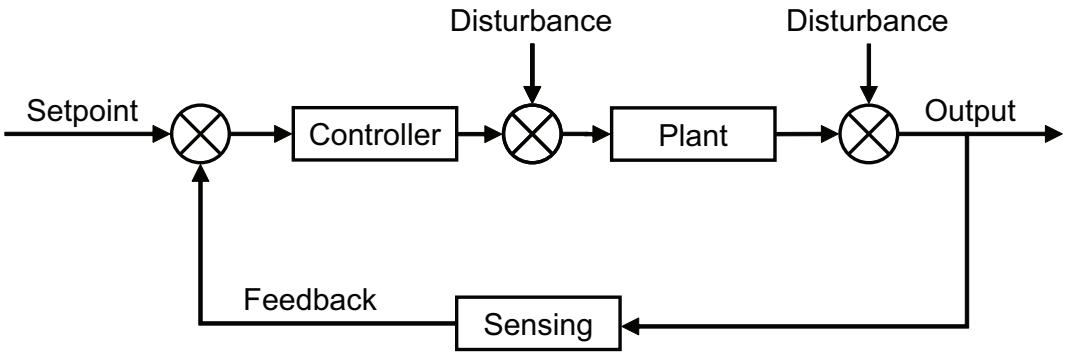


Figure 1.1: The general structure of an FCS, showing the influence of controller decisions upon the output from the plant.

FCS: the Controller, Plant, and Sensing blocks (Nise 2004, chap.1). The basic operation is that a setpoint is specified, and the plant is controlled in an attempt to influence the output to match this setpoint. Sensing is used to measure the output, and the controller's decisions are made on the basis of some function of the feedback and setpoint.

An FCS is operational from the moment the setpoint is first specified, and continues until its overall objectives are met. The controller can be considered as a decision making process because it directly influences the plant, which produces the output. In many cases the frequency with which the feedback control loop can execute affects the performance of the system, especially if the setpoint is variable or disturbances are present. In these cases, the controller can be considered as an ADM process because the time taken to make each control decision influences the control frequency.

Consider now the timing of these data flows. Figure 1.2 visualises the same data flows from the general FCS in Figure 1.1, but shows them in a Unified Modeling Language (UML) timing diagram. The three blocks from the FCS are shown as parallel processes, with their state indicated by the solid black lines transitioning between the relevant labels for each process. The grey arrows between processes indicate when sensor data is processed and available for the controller, and when the controller's decisions are calculated and available to induce actions in the plant.

The diagram shows that there are finite time periods required to process sensory information and determine controller decisions given this processed information. These time periods are labeled t_S and t_C respectively. It is important to note that control systems are typically

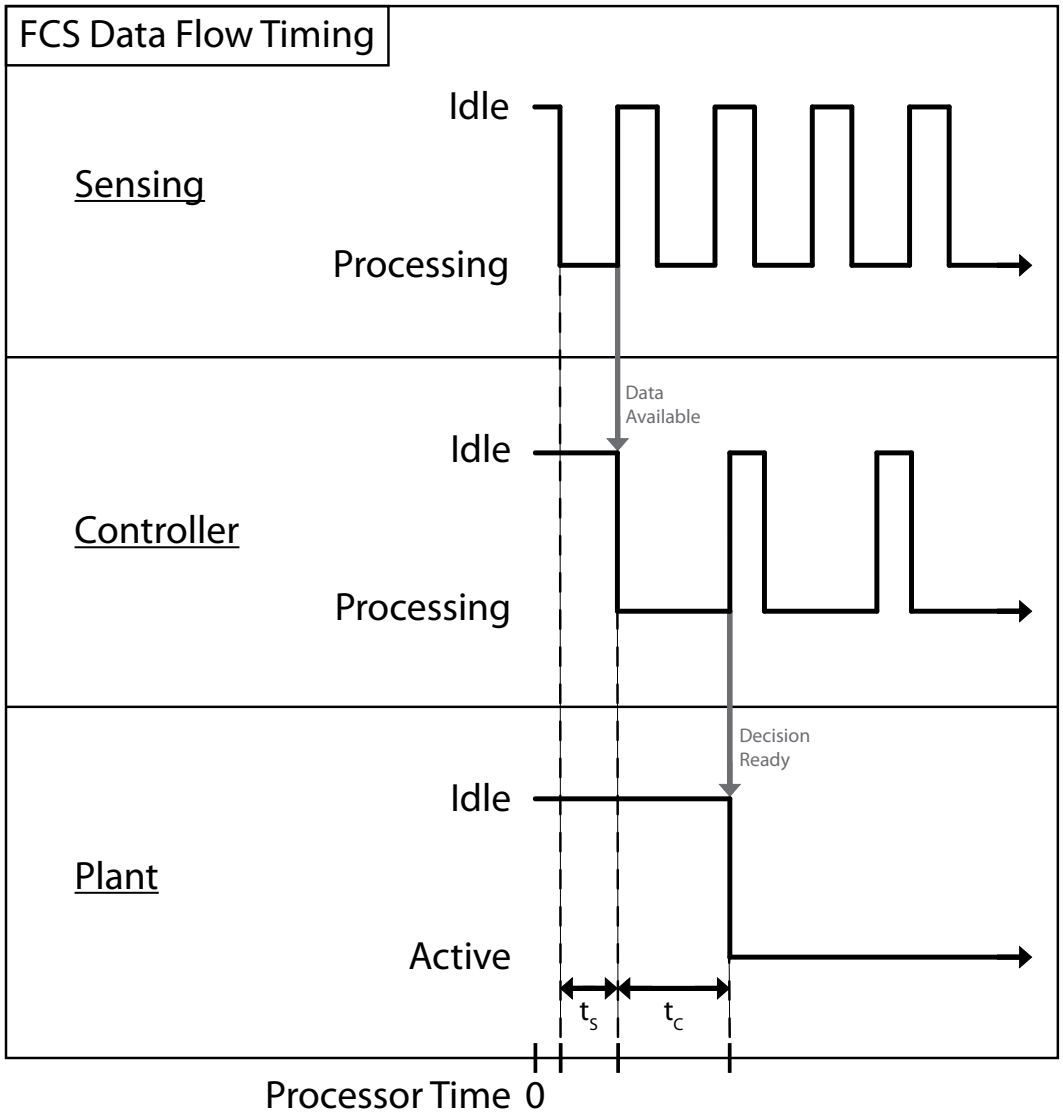


Figure 1.2: UML timing diagram for a basic FCS, showing the relationship between processing time and the dynamics of the state space.
The diagram is simplified for readability, and it is important to note that in general the transitions between the idle and processing states for the sensing and controller blocks may not be as regular as shown. In most practical systems sensing is effectively continuous and the processing occurs over batches of observations. Similarly, control is also likely to be continuous, with the incoming processed sensor data buffered and processed by the controller in batches. The processing time required by the sensing and controller blocks is also likely to vary depending on the information to be processed.

modeled and analysed in the continuous time domain, where these discrete time periods are not considered. It is only when the system is modeled in a discrete time domain that they become apparent and important. In this thesis, the processing performed by decision making systems is considered in such a manner because it is performed by physical computational hardware, which at its most fundamental level operates in discrete units of time. The main point to draw from the diagram is that the result of the discrete processing stages is that any action taken by the plant is made on the basis of the state of the world as it was at least $t_S + t_C$ worth of time ago.

There are many situations in which this point is unimportant. For example, in a static state space the plant's actions remain valid despite the delay. Alternatively, if t_S and t_C are sufficiently low then the effect of this delay may be insignificant. In other cases, the measurable performance of the system is related only to the the degree of completion of its goals, and not its time response. This thesis however is motivated by those situations in which this effect is significant: dynamic state spaces that are sufficiently complex that t_C affects the system response, and where system performance is principally measured by the time taken to achieve the objectives. Such situations are examples of TOADM problems.

The proposed structure of solutions to TOADM problems in this thesis considers the general ADM process itself as an FCS. This is illustrated by Figure 1.3, in which the original system from Figure 1.1 is subsumed by the ADM process. Relevant information about the output of the system is collected by the Measurement block, which provides feedback to the ADM block for subsequent decisions (and may also store the information for later use). These decisions are the input to the original FCS and may comprise either an adjustment to its setpoint, or instructions that affect the operation of its controller, or both. In either case the decisions are put into effect by way of adjustments to the system's parameters.

These ideas are further developed in later chapters of this thesis, showing how the measurement component can incorporate timing information in order to convert this structure into the full TOADM process. The effect is an investment of some small amount of time in order to determine the set of system parameters that are expected to minimise the remaining time required to complete the objectives. If the amount of time saved by adjusting the system parameters is greater than the amount of invested time, then the strategy is capable of reducing the total time required to complete the system's objectives.

These ideas provoke the following questions which motivate much of the work described in

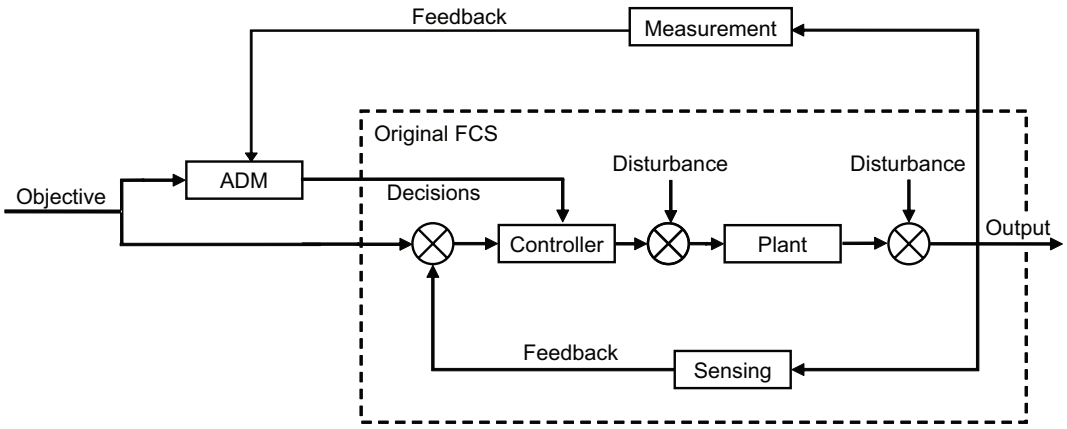


Figure 1.3: The general structure of an ADM control system, showing that measurement of the system's output is fed back to the ADM such that subsequent decisions can attempt to further optimise performance.

this thesis:

- What is the best use of limited computational resources?
- How can an ADM system make use of this knowledge?
- How precise and timely must this knowledge be, in order to be useful?

This thesis addresses these questions, and the TOADM problem structure and solution processes described in subsequent chapters provide a suitable framework to answer them.

There are many example applications which fit within the TOADM problem domain, or can be posed in a such a way that the TOADM structure can aid their solution. These include scheduling and search problems, turn-by-turn navigation software, infrastructure or personnel management, control of autonomous robots, and many topics involving data analysis in fields such as finance, intelligence, or defence. The key commonality between these disparate areas is that their applications involve committing significant resources to the act of decision making. For example, scheduling or infrastructure management require analysis of enormous combinations of options, search requires finding particular items in massive datasets, and turn-by-turn navigation requires planning over road maps for entire countries. Some of these examples are considered further in Chapter 6, however the main example throughout this thesis is the control of autonomous robots.

This section has described the TOADM problem and shown how it can be considered as the need to account for the timing of data flows throughout a system. The remainder of this chapter articulates the main thesis contributions and describes the structure of this document. Overall, this thesis aims to validate the following hypothesis:

That the TOADM process is able to meet or improve upon the performance of any method that uses only fixed system parameters, given the same sources of state information.

1.2 Thesis Contributions

This thesis introduces a new problem domain known as TOADM, and presents a process that can provide practical solutions within it. However, the main focus of the thesis and its principal example is a sub-problem within the TOADM domain, known as the Time-Optimal Planning and Execution (TOPE) problem (described in Section 2.2.1). The main contributions involve the TOPE process, by which solutions to this problem can be found. The TOPE process provides a means of reducing the total time required to achieve a goal, compared to any planning system which only specifies tuning parameters at design time.

The principal contributions of this thesis are:

- A derivation of the equations governing the TOPE problem, showing that it can be considered as a form of dynamic programming.
- Validation of the TOPE process via simulation showing that when run sufficiently quickly and accurately, it can significantly improve upon the performance of comparable methods from the literature, even when these methods use the best possible fixed parameters given ground truth values.
- Implementation of heuristic and statistical TOPE parameter estimation techniques for Unmanned Ground Vehicle (UGV) planning scenarios, showing that many different practical implementations lie within the operating region in which they outperform alternative methods.
- A Monte-Carlo analysis of various TOPE parameter estimation techniques in randomised scenarios, showing that the best performing techniques have 'out-of-the-box'

performance that significantly improves upon comparable methods from the literature, without requiring any prior knowledge of the scenario or any parameter tuning.

In addition to the derivation of the TOPE problem and its solution, other contributions of this thesis are:

- The definition of 'time-focussed metrics' as a means by which the time taken to make decisions and the performance of a system given these decisions can be compared.
- The identification of a collection of trade-offs between computation and execution time for a range of planning algorithms, data structures, and applications.
- A method to determine an admissible bound on the cost of the optimum plan for the class of 'anytime algorithms', and a technique which uses this bound to reduce computation time in static planning scenarios.
- A procedure to determine the theoretical minimum time to achieve a goal for any given scenario and its available tuning parameters.
- Procedures to assess the utility of a particular tuning parameter for the TOPE process, and the sensitivity of the overall results to inaccuracy in its tuning.

1.3 Document Structure

This chapter has introduced and motivated the TOADM problem, and described the main contributions of this thesis. It was suggested that the TOPE problem lies within this domain, and it will be shown that solutions to this can improve the performance of UGV planning systems. The following chapters describe the TOPE process used to solve this problem, by way of continually adjusting tuning parameters to a planning system such that the total time required to achieve a goal is minimised.

Throughout Chapters 2 through 5, the content is presented with reference to the TOPE problem, rather than the larger TOADM domain. This is intended to aid understanding of the material, since TOADM represents an abstract class of problems, whereas the TOPE problem is a concrete example from a well-studied field, and can be directly applied to UGV

path planning. In Chapter 6, the TOPE-specific concepts, processes, and techniques developed in the preceeding chapters are all shown to be examples of more generic formulations that are then applicable to every instance of a TOADM problem.

Chapter 2 presents background material and related work in the planning field. This material is presented with its historical context, showing that much of this work focussed on reducing either the computation time required, or the 'cost' of the output plans with respect to the optimum. Subsequent work enabled the trade-off between these two concepts to be manually, and later automatically, adjusted. Various cost metrics are described, and several fielded systems that perform trade-offs in replanning scenarios are assessed. It is shown that only one class of metric enables a practical trade-off between the two concepts, and requires that the cost of plans be measured in terms of execution time. Finally, it is shown that this concept can be extended to ADM problems, and enables the creation of TOADM systems.

Chapter 3 builds upon this result, and identifies parameters that enable effective trade-offs between computation and execution time for planning systems. Known techniques for reducing the overall time required to achieve a goal for planning systems are described and assessed. These include a technique which exploits parallelisation of the levels in a hierarchical planning system, and which is shown to reduce the overall execution time required. All of the techniques and system parameters discussed represent means of trading between planning and execution time, and a series of experiments are performed to assess the effect of each trade-off. These results are specific to the UGV planning example but the concept is generic, since it is suggested that almost every ADM system exhibits such tuning parameters. Given the existence of such parameters, they can be used as the leverage to improve performance in TOADM systems.

Chapter 4 presents the TOPE problem in detail, derives the equations which govern it, and describes a process to solve the problem in dynamic environments. It is shown that this process is capable of simultaneously reducing both the computation and execution time required, provided it can be performed sufficiently fast and accurately. The process involves the estimation of tuning parameters for the planning system that affect the trade-off between this computation and execution time. A procedure is developed to assess the sensitivity of the process to the accuracy and timeliness of the estimation. Examples are given which validate the process and show that the sensitivity is low enough to allow

practical implementations to be effective. The procedures are formulated for the TOPE problem, but the underlying ideas are applicable to any TOADM system.

Chapter 5 presents practical implementations of the TOPE process, and starts by describing the two main classes of estimation techniques. Several examples of each of these classes are implemented and used to validate the TOPE process by comparative simulation with alternative planning systems from the literature, using a common planning scenario. It is demonstrated that these TOPE estimators are capable of improving upon the performance of even systems which use the optimum set of fixed tuning parameters – a result which confirms the original thesis statement. A Monte-Carlo analysis comparing TOPE techniques against existing techniques in the literature confirms that the average case performance of the best TOPE techniques is superior to existing techniques in a wide range of scenarios. In addition, the implementations with the best performance are from a generally applicable class of techniques, meaning they can be expected to perform well for other TOPE scenarios, and not just for UGV planning systems. Like the procedures developed in Chapter 4, the principles used by each technique are also applicable to TOADM systems.

Chapter 6 presents conclusions, summarises the main contributions of this thesis, and discusses areas of future work in this field. Above all, this chapter re-examines the TOADM problem and shows that many of the results obtained for the TOPE problem are generally applicable.

Chapter 2

Background and Related Work

This chapter presents a review of literature from the fields of computer science, mathematics, and engineering, all of which have made significant contributions to the study of planning. Paraphrasing LaValle (2006, chap. 1), planning can be defined as the process of computing a sequence of actions that, when executed, achieve a goal. There are many domains in which planning is a useful tool, but it is most important for those in which the act of executing the computed plan has a bearing on the performance of the task. This focus means that topics such as scheduling and search are not discussed herein, since the use of planning for these problems is typically to produce an answer to some question, and the execution of the plan is not necessarily relevant. In particular, this chapter considers the role of an agent of execution; an entity which performs the sequence of actions given by the plan.

Section 2.1 presents a brief overview of the field of complexity analysis, in order to define the concept of computational intractability. Section 2.2 then presents the fundamental planning problem, and shows that it is an optimisation over all possible plans, which is computationally intractable for almost all practical problems. As a result, approximations must be made to solve it, and these are typically applied at either the data or algorithm levels, or both. This section also presents the principal application and example scenario used throughout this thesis.

Section 2.3 discusses the two main problem domains for agent-based planning systems, and the data structures that are used to represent them. Section 2.4 then describes the types of algorithms that are employed for each of these domains. Section 2.5 describes replanning systems, which make use of these algorithms and data structures in dynamic

domains where a single *a priori* plan is insufficient to achieve a goal. Section 2.6 presents metrics that are used to assess the performance of the algorithms and systems described in preceding sections. This final section also shows that the type of metric required to implement solutions to the TOPE problem is the same as those required to implement TOADM systems. This type of metric is denoted *time-focussed*, and a simple formulation for use in the UGV planning scenario is presented.

As a final note, this chapter does not explicitly define all relevant planning terminology. A suitable reference for this purpose is LaValle (2006), and any required notation will follow its definitions unless otherwise specified. Three other useful references that cover the planning field are Cormen et al. (2001), Russell & Norvig (2003), and Kavraki & LaValle (2008).

2.1 Complexity Analysis

The principal motivation for solving the TOADM problem is to provide a consistent approach to problems in which limited computational resources act as a design constraint. Examples of such problems are common in many domains, and the class of problems itself is studied by the field of complexity analysis (Papadimitriou 1994). Decision making problems are typically analysed by asking whether the outcome of the decision was a system response better than some threshold value, as the original problem of finding the optimum decision can be no easier than this (Tsitsiklis 1984, Matthews 2008).

When a system's future state can be determined entirely from its current state it is said to be *Markovian* (Bellman 1957*b*), and it was shown in Papadimitriou & Tsitsiklis (1987) that Markov Decision Processes (MDPs) lie within complexity class P. This class of problems can be solved in polynomial time on a universal Turing machine, meaning their runtime is bounded by S^K, where S is the size of the problem, and K is some constant (Papadimitriou 1994, Matthews 2008). The control theoretic formulation of the TOADM problem however, is not an MDP since the allowance for disturbances between the controller and plant, and upon the output due to the dynamic state space, means that the system is only partially observable. Such a Partially Observable Markov Decision Process (POMDP) lies within the complexity class PSPACE (Papadimitriou & Tsitsiklis 1987). This class of problems can be solved in polynomial space, meaning that its memory usage is bounded by S^L, where S is again the size of the problem, and L is some constant.

Furthermore, the complexity class PSPACE contains the class NP, wherein problems can be solved in polynomial time only on a nondeterministic Turing machine. Such a machine has the ability to guess a solution, and to verify whether it is the optimal solution in polynomial time. This implies however that many such guesses (up to S of them) may be required in order to find the optimal solution, and as a result the class NP and thus all superclasses are termed *intractable* since their runtime grows at a greater than polynomial rate as the size of the problem increases. Most NP-hard problems can only be solved in real-time, or some practical amount of time, for small values of S.

2.2 The Planning Problem

All planning techniques involve the computation of sequences of actions that achieve a goal (LaValle 2006, Russell & Norvig 2003), where an action is a means of transitioning from one state to another. This computation always involves exploration of actions within some set of possibilities, followed by or in parallel with analysis that determines the appropriate choice of a sequence of these actions. Some planners have a bounded set of possibilities, and can explore the entire set and choose the optimal sequence of actions, given a function with which to compare sequences. Others are unbounded and must determine when a given sequence of actions is sufficiently good, given some decision making process to do so. In both cases, the planning process can be considered as a form of optimisation; the first case assesses every possibility and selects the optimum, and the second case continues the optimisation until a condition is met.

This optimisation is computationally intractable for almost any practical problem. There are many possible reasons for this, but in general it is that there exists at least one continuous variable that affects the possible action sequences. Any continuous variable, whether bounded or unbounded, can take an infinite number of possible values. This leads to an infinite number of possible action sequences, and thus the full optimisation requires infinite time to assess them all. Even in discrete state spaces, it is shown in Canny (1988) that both the generalised mover's problem and single-source single-destination planning problem are NP-hard.

To resolve this intractability, approximations are made to render the problem manageable, and are typically applied at the data or algorithm levels. At the data level, these approx-

imations often require the discretisation of spatially continuous variables; for example, the partitioning of a continuous space into cells with a discrete data structure such as a grid. Alternatively, although a variable is spatially continuous, it may be sufficient to know its value relative to some threshold, and to filter the information into a discrete-valued representation. At the algorithm level, discretisation is less useful, since the same effect can be achieved earlier at the data level. As a result, algorithmic approximations typically take the form of rationalisations; means of justifying whether a technique is sufficient to meet the overall objectives of the problem scenario. For example, users of graph search algorithms will frequently forgo optimality guarantees in order to reduce the time taken by the planning process (Pohl 1970, Pearl 1984, Thayer & Ruml 2008). This is justified if the overall objective benefits from this reduction and is not harmed by the loss of optimality.

The general difficulty in solving the planning problem exactly is that there is insufficient time available to perform the full optimisation. The solution taken by all the literature reviewed in this chapter is to make approximations and trade-offs such that the overall objectives of the problem scenario are still met, yet the computational requirements of the problem are reduced. This thesis is then motivated by the desire to somehow optimise the choice of these approximations and trade-offs, and thus meet the overall objectives in the most timely manner.

2.2.1 Principal Application and Example Scenarios

There are many fields and applications in which the planning problem must be solved, including autonomous vehicle navigation, Global Positioning System (GPS) navigation software, automated share trading, computer games, and search algorithms. All involve problem domains in which significant computational effort is required, and for which the primary performance metric is the total time required before a goal or goals are achieved. As its principal application and example however, this thesis discusses the design of a planning system for use on an UGV which shares this performance objective.

Figures 2.1 and 2.2 illustrate typical scenarios for this application, wherein the vehicle must travel from its starting position to a goal location. The total mission duration is measured from the time the goal location is specified to the time the vehicle reaches it. Provided that safety of the vehicle is ensured (by avoiding collision with obstacles and the traversal

of unsafe regions), the vehicle is allowed to take any route through the intervening space. The objective is simply to reach the goal safely, but the performance metric is the mission duration, which should be minimised.

In the figures, the start and goal of the search are marked by the indicators 0 and 1, respectively. The gray-scale overlay represents a 'cost' denoted c, which ranges from highly traversable black regions ($c = 0$) to completely untraversable white regions ($c = 1$). That is, the brighter the pixel, the higher the cost. This information is a filter over raw 3D point clouds that indicate the locations of laser-reflective objects in the environment. These point clouds were obtained by equipping the vehicle with two SICK 2D scanning laser range finders in a 'push-broom' configuration, then driving at a slow speed within the testing area to obtain a very dense dataset[1]. A vehicle-specific filter is applied, which assess the slope of the terrain over multiple resolutions, and also the range of heights in a given region. Steep regions and solid obstacles are then marked as highly untraversable areas, whereas flatter areas are more traversable. The cost values are stored as 32-bit floating point numbers, ranging between zero and one. More detail on the construction of these 'cost maps' is available in Underwood (2009).

This cost information is stored in a runtime configurable data structure where the value of each region (or point if the structure is continuous) is the maximum cost value within its bounds; i.e. a conservative filter. Figure 2.3 shows the effect of storing this information in regular grids of various cell sizes, using square cells with side lengths ranging from $0.5m$ to $8.0m$. The main feature to note from this figure is that as the grid cell size increases, the proportion of high cost regions in the representation increases significantly. It will be shown in Chapter 3 that although it is far faster to compute plans through a coarser grid representation, the expected execution time increases significantly.

Since the performance metric for these scenarios is the time taken to achieve a goal, the cost information is further filtered to give a maximum velocity limit proportional to its cost value. The maximum velocity limit for a point or region is given by

$$v = p \times v_{max} + (1 - p) \times v_{min}$$

where $p = (1 - c)^n$, $n = 10$, $v_{max} = 5m/s$, and $v_{min} = 0.5m/s$. The particular choice of

[1]This raw dataset is far higher resolution than any of the cost maps generated from it.

Figure 2.1: The longer 2D path planning scenario employed in many of the experiments in this thesis. The background aerial photo shows the underlying terrain in an autonomous vehicle testing area located in Marulan, Australia. The gray-scale overlay represents information available to an agent operating in this scenario, and is a filtered 'cost map' that reflects the difficulty of traversing regions of this terrain for a particular agent. The agent is required to travel from the red indicator marked 0 to the other indicator marked 1, and is desired to do so in the shortest possible time. The distance between these indicators is $638m$. The white rectangle bounds the area used in the shorter scenario shown in Figure 2.2.

Figure 2.2: The shorter 2D path planning scenario employed in many of the experiments in this thesis. The distance between the indicators is $124m$.

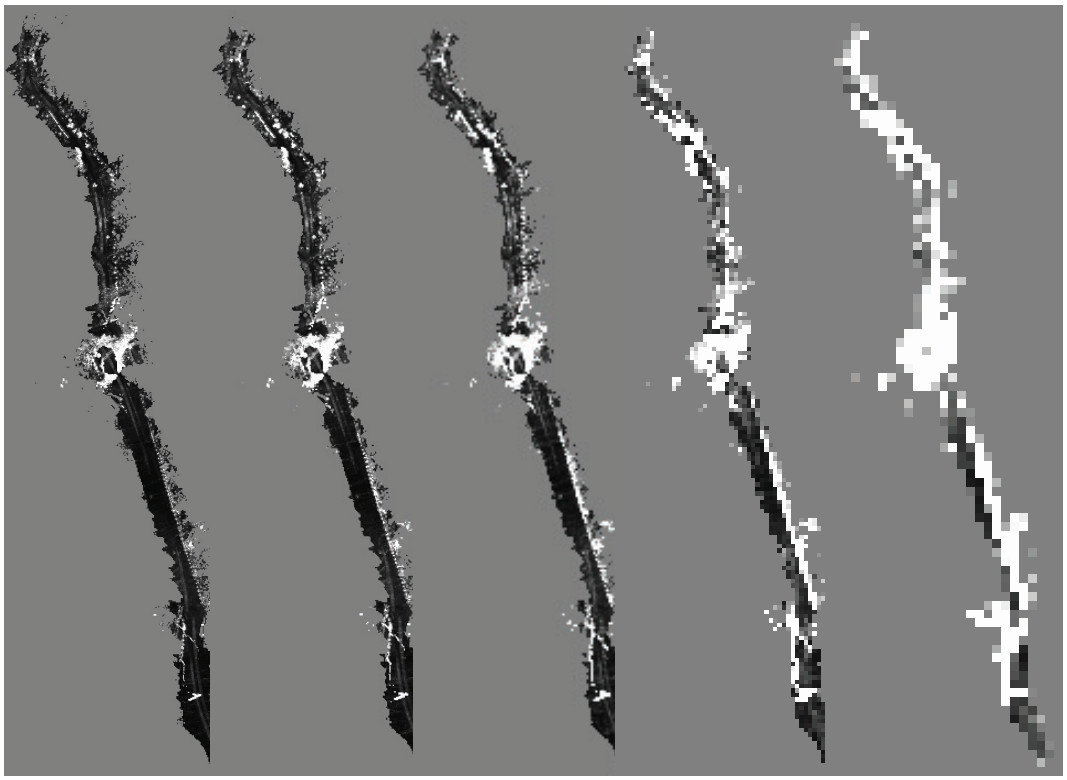


Figure 2.3: A comparison of cost information as stored in regular grids of increasing cell size. Since the value of each cell is equal to the maximum of the cost information within its bounds, as the cell size increases, the representation of the terrain appears less traversable. The grids shown use square cells of side length $0.5m$, $1.0m$, $2.0m$, $4.0m$, and $8.0m$, shown from left to right.

a degree-10 polynomial is arbitrary, but was empirically determined as a suitable value in previous work on the CAS Outdoor Research Demonstrator (CORD) vehicles.

This operation converts the cost map into a *mobility map*, in which the information specifies the ease of mobility in a given region (Karumanchi 2010). It was shown in Karumanchi et al. (2009, 2010) that the use of such mobility information can improve the performance of UGV systems beyond that achievable without it.

Section 2.6.2 provides a further motivation for the use of mobility information, showing that it allows the expected traversal time through a region to be easily determined, which is required for TOPE systems to operate effectively. The distance across each region (in the direction of travel) divided by the velocity limit determines the expected traversal time for the vehicle, and similarly for continuous data structures where the integral over each region

is used. Further discussion of the real-time construction of these cost maps is presented in Section 4.3.2.

UGV planning systems that operate in scenarios such as this fit within the TOADM problem domain if the state space in which they operate is dynamic, and time-optimal performance is desired. In this example even if the underlying environment is static, the cost map is built up incrementally as the vehicle's sensors perceive the environment, meaning that the planner's state space is dynamic anyway[2]. For a planning system to obtain time-optimal performance in a dynamic state space then requires a trade-off between the time spent planning and the quality of the plans, in order to adequately respond to the dynamics.

TOPE is a subproblem within the TOADM domain, which attempts to determine the appropriate decisions to optimise this trade-off. The name should be read as time-optimal *planning and execution*, with the emphasis implying that the combination of planning and execution is to be time-optimal. This is to distinguish it from a related field called 'time-optimal planning', which is further discussed in Chapter 3. The TOPE problem is the main focus of this thesis partly because it directly relates to the principal application of improving UGV performance, but also because it covers all the relevant aspects of the TOADM domain.

Solutions to the TOPE problem are necessary for at least two reasons. Firstly, because time is an important measure of performance for many planning systems, and especially so for any that operate in dynamic state spaces. Secondly, since the motion planning problem is known to be at least NP-hard (Canny 1988), existing techniques invariably require trade-offs to reduce computation time at expense of solution quality. Evidence for both of these assertions is found in the literature review and background material below, and subsequent chapters illustrate that the TOPE problem can be solved by way of a process which makes these trade-offs in a time-optimal manner.

2.3 Data Structures Used in Planning Domains

The purpose of this section is to describe the two main domains in which planning is used, and the types of data structures that are useful for storing the state space information in

[2]While there are some circumstances in which the differences between a fully dynamic environment and one which is incrementally perceived are relevant, for the majority of this thesis the distinction is unnecessary.

each. This state space, X, is a set of individual states, x, which are vectors of relevant features. The two main domains used to represent this state space are 'discrete'; where the space is a collection of distinct states, and 'continuous'; where any point in the space has an associated state. In both cases, data structures are used to represent the underlying data as maintaining and/or accessing the raw data would be inefficient in terms of memory or access time. Despite the benefits of maintaining raw data in perpetuity, this is usually impractical for real systems due to the storage requirements and overhead of filtering data into a usable form at runtime (Grover 2007). Discretisation is also often applied to the data values as well as the spatial dimension, a technique known as discrete quantification.

These two domain concepts are illustrated by Figure 2.4, which shows two representations of the data values shown in black. The blue curve is a continuous representation fitting the function with least squared error to the data, and the red line segments are a discrete form which takes the average value in 10 equal sized bins of the spatial dimension.

To use either domain, the data values are filtered into the relevant data structure, and then only this representation is maintained. For the continuous representation, only the parameters of the functional fit (or other relevant information depending on the technique) are stored. For the discrete, only the filtered value of each bin is recorded. To subsequently determine the data value at a given spatial value requires evaluating the function at this spatial value, or returning the value associated with the appropriate bin, respectively.

2.3.1 Discrete Domains

Discrete domains make some form of approximation of the true planning problem, such that it is solvable by the planning algorithm, and the results are appropriate. A discrete domain does not necessarily imply a finite state space, but it must be countable to provide certain guarantees needed for some algorithms. In many cases, the size of the state space is too large to store in its entirety, thus many representations use an 'implicit encoding', meaning that they can generate states from a set of rules and store only a relevant subset of the space.

The reasons for employing a discrete domain are varied, but typically are either that the computation required to find a plan would otherwise be prohibitive, or that the quantity of state space data is impractical to maintain in full. Discrete domains solve both these

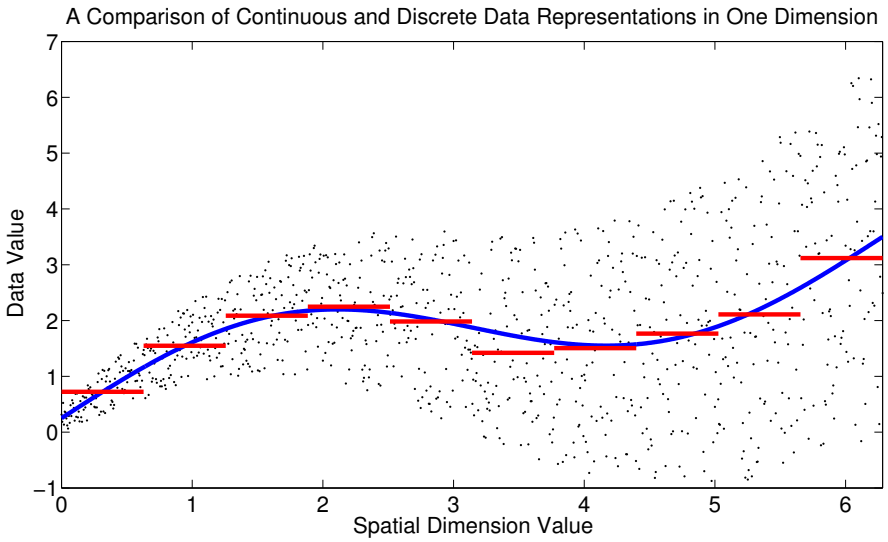


Figure 2.4: A comparison of continuous and discrete data representations in one dimension. Each data point represents a sample made at a particular spatial location, resulting in a particular data value at this location. The discrete representation takes the average data value for all data points within a finite spatial range. The continuous representation fits the cubic spline with minimum least squared error to the entire data series.

issues because the limited number of states restricts the number of possible plans, and simultaneously encapsulates or implicitly encodes the state space data.

Examples of data structures used to represent discrete domains include:

Occupancy Grids An array of regularly sized regions of the state space, which apply some function to all the data within the region. Typically an averaging or threshold function is used to generate a single value, but in general any data can be maintained (Moravec & Elfes 1984, Allen et al. 2009, Saitoh et al. 2010).

2^N-trees A tree structure where each node holds a region of information like a grid, but the length of each of the N sides in this region is $\frac{1}{2^d}$, where d is the depth of the node in the tree. Each region has a maximum capacity of data points, and splits to form 2^N new descendant nodes if this capacity is exceeded. Known as quad-trees and oct-trees in two and three dimensional spaces respectively, these data structures are particularly memory-efficient for sparse data (Samet 1982, 1988, Yahja, Singh & Stentz 1998).

kd-trees A multi-dimensional form of a binary tree (Friedman et al. 1977). Data points are first sorted by their first dimension, and divided into two sets about the median value. For each set, the points are further sorted by their second dimension, and further subdivided about the median of this dimension. This proceeds for all dimensions, then recurses into each subdivision and repeats the process for all dimensions. The process completes when there is no subdivision with more than some maximum capacity of data points. The tree can be constructed in $O(n \log k)$ time and the nearest subdivision in the tree to a given data point can be found in $O(\log k)$ time (LaValle 2006, chap. 5).

Cell Decompositions A division of the data into irregular regions, using some fixed procedure. A function is then applied to the data in a given region, leaving a single data value for each. Examples include various polygonal tessellations (Cormen et al. 2001) and Voronoi decompositions (Dirichlet 1850, Voronoi 1907). Occupancy grids are often considered to be a special form of cell decompositions.

Graphs A set of 'vertices' connected by 'edges', where either or both elements can hold data. There are numerous classes of graphs defined by the structure of the edges and connectedness of the vertices, and many algorithms exist to perform efficient operations upon them, including planning. Graphs can also encapsulate other data structures since they require only a list of vertices and their connected edges. For example, all the above structures can be represented by graphs where the regions are vertices and their neighbouring regions provide a list of edges (Russell & Norvig 2003).

2.3.2 Continuous Domains

Continuous domains fit a function to or operate upon the state space data, such that information is available anywhere in the space, rather than averaged over discrete regions or only at the data points. Since the representation is continuous, planning algorithms which operate in these domains cannot guarantee to find a solution, even if one exists (LaValle 2006), and instead offer alternative guarantees which are often sufficient for their target applications. Continuous domains suit a large number of applications, but are most commonly used in high dimensional spaces, where a discrete domain would suffer from an exponential growth in the number of states.

Examples of techniques that fit a function whose parameters are a representation of a continuous domains include:

Interpolation A technique to provide additional data at sampled points within the bounds of the discrete set of known data points (Stewart 1999). The simplest technique is to locate the nearest data point to the sample and return the same value, known as piecewise constant value interpolation. Linear interpolation improves upon this, returning the value of the sample's location along the line between the nearest two data points. Polynomial fits are similar but replace the linear interpolant with a polynomial (of maximum degree $N-1$ given N data points to prevent over-fitting). Splines fit low degree (usually cubic) polynomials between each pair of data points, but ensure the second derivatives at the point between two pairs are equal.

Least Squares A regression technique that applies any of the preceding interpolation techniques, but removes the constraint that the fit must pass through each data point. A least squares fit minimizes the sum of squared residuals, where a residual is the difference between a data point and the value of the functional fit at that point (Gauss 1809, Phipps & Quine 1998, Stewart 1999).

Gaussian Process Regression (also known as Kriging, Kolmogorov Wiener prediction, or best linear unbiased prediction) A form of Bayesian inference which fits a multivariate Gaussian distribution to the data points (Rasmussen & Williams 2006). The Bayesian aspect is that the initial data points are used as a prior (under the assumption that their values are normally distributed). Given a likelihood function for the observations, the value at any new sample can be obtained by combining the prior with the likelihood, as per Bayes' rule. A significant benefit of this technique over others is that each sample has an associated variance that can be interpreted as a measure of the confidence of the functional fit at its location (Vasudevan et al. 2009, Karumanchi et al. 2009).

2.4 Planning Algorithms

This section describes the most prevalent planning algorithms used in discrete and continuous domains. The purpose of this discussion is to show the chronological development of

planning algorithms, with particular emphasis on the effect each has on the planning time required, and the *cost* of the solutions with respect to the global optimum.

These costs are the output of a cumulative function over each action in the sequence of actions that comprise a plan, such that a plan with a lower cost than another can be said to be qualitatively better. As an example, a common cost function used in 2D path planning applications is the sum of the distance travelled by the agent when executing each action in the plan. The *global optimum* cost is then the minimum cost possible given the state space and action space for the agent (and thus can change with time). In most cases the global optimum cost cannot be known ahead of time, and can only be determined by completion of the full planning optimisation, which is typically intractable. As a result, many of the algorithms described in this section employ an *admissible heuristic* – a cost function guaranteed never to over-estimate the true global optimum – to justify when there are no alternative action sequences better than a particular one.

2.4.1 Discrete Algorithms

Discrete algorithms operate in a discrete domain, and find a plan by *visiting* states in this domain and assessing their suitability to achieve a goal. A requirement for a discrete algorithm is that it be *systematic*, meaning that in the limit as the number of iterations tends to infinity, the algorithm will assess every achievable state. For a finite state space, this provides a guarantee of *completeness*, meaning that a solution will be found in finite time if one exists, which is desirable for many applications. For an infinite state space, certain systematic algorithms will still guarantee to find a solution in finite time if it exists, but may continue searching forever if no solution exists. Other algorithms (such as Depth First Search) may have no such guarantees. Although there are not necessarily any bounds on the time required to compute a solution, they can often be determined if the domain is finite, and the number of states is known.

LaValle (2006) provides a template (depicted in Algorithm 2.1) for discrete planning algorithms and describes several, including; Breadth First Search, Depth First Search, Dijkstra's Algorithm, Best First Search, and A^*. In this procedure, Q is a sorted container of states, initially populated with the starting state for the search, x_I. States are marked as 'visited' (usually implemented as a separate container, with direct and fast access to elements) when

they are first added to Q. The algorithm proceeds by extracting the first state in Q, failing if there are none, and succeeding if this state is the goal state, x_G. In all other cases, every action that can be taken from the extracted state, x, is assessed. The neighbouring state, x', is marked as visited and added to Q, with the ability to perform additional operations if x' is already visited. Actions, u, are drawn from the *action space*, U, with $U(x)$ giving the subset of actions in this space that are available from state x. A *state transition function*, $f(x, u)$, determines the future state, x', obtained by applying action u in state x.

Algorithm 2.1 A general template for forward search (after LaValle (2006, chap. 2)).

```
1: Q.insert(x_I) and mark x_I as visited
2: while Q ≠ ∅ do
3:    x ← Q.GetFirst()
4:    if x = x_G then
5:       return SUCCESS
6:    end if
7:    for all u ∈ U(x) do
8:       x' ← f(x, u)
9:       if x' ∉ visited then
10:         Mark x' as visited
11:         Q.insert(x')
12:      else
13:         Resolve duplicate x'
14:      end if
15:   end for
16: end while
17: return FAILURE
```

Particular forwards search algorithms can be instantiated by specifying the sorting criteria of Q, and sequences of actions from start to goal can be found by augmenting each state, x', with a marker noting its parent state, x. For example, Breadth First Search implements Q as a First-In, First-Out queue, Depth First Search implements it as a Last-In, First-Out stack. Dijkstra's Algorithm sorts Q by the *cost-to-come*; the sum of the action costs between each pair of states in the trail of states from a particular x in Q back to the x_G. The A^* algorithm sorts Q by the sum of the cost-to-come and the expected *cost-to-go*, as determined by a function known as a *heuristic*.

LaValle (2006) and many other works describe these algorithms in much greater detail. As a result, they are not described further here, except to illustrate their different best and worst case planning times and costs. In many cases, the cost of a solution using a

discrete algorithm can be provably optimal with respect to the state space (which itself is discrete and thus suboptimal). This is achieved by noting the dynamic programming principle, namely that any subset of the set of actions in an optimal plan is also optimal. A technique called *value iteration* then allows the algorithm to store the optimal sub-plan to any particular state, iteratively building up the plans until the goal is achieved. Since the plans are built out of provably optimal sub-plans, the full plan to the goal is optimal. Again, LaValle (2006, chap. 2) has a thorough description, derivation, and proof of these concepts.

For the A^* algorithm, for example, it is shown in Hart et al. (1968) and Pearl (1984) that optimality can be guaranteed if the heuristic function used is *admissible*. An admissible heuristic is one which is guaranteed to equal or under-estimate the true cost of the optimal plan between two states. The A^* algorithm uses the heuristic to restrict the number of states that must be evaluated to find the optimum solution plan, by only evaluating those states whose cost-to-come plus heuristic cost-to-go are lower than their neighbouring states. The A^* algorithm in particular is guaranteed to expand an equal number or fewer states than any other algorithm using the same heuristic (Pearl 1984), meaning it is provably the best choice of optimal heuristic-guided graph search algorithm, provided that a suitable heuristic function is available.

By following the historical progression of discrete algorithms in the literature, a trend to reduce the total time required to achieve a goal can be observed. The introduction of complete algorithms such as Dijkstra's Algorithm in 1959 (Dijkstra 1959) and Breadth First and Depth First Searches in the 1960s and 1970s (Dreyfus 1969, Cormen et al. 2001, Russell & Norvig 2003) allowed solution costs to be minimised, but the required planning time was typically high, especially in the worst case. Heuristic algorithms such as Best First Search (Dreyfus 1969, Pearl 1984, Russell & Norvig 2003) and A^* (Hart et al. 1968, 1972) were developed in the 1970s and 1980s, and were able to dramatically reduce this planning time in the average case without raising it in the worst case, by making use of domain specific assumptions (LaValle 2006, chap. 2). The Weighted A^* algorithm (Pohl 1970, Pearl 1984) uses a deliberately inadmissible heuristic to attempt to further reduce the planning time, at the expense of a higher upper bound on the solution cost of the plans.

For planning in discrete domains where previous results can be repaired rather than starting again from nothing, the use of incremental algorithms in the 1990s and early 2000s, such as

D^* (Stentz 1993, 1994), Focussed D^* (Stentz 1995), and $D^* - Lite$ (Koenig & Likhachev 2002, 2005) were able to reduce the planning time for these repairs, without significantly affecting the initial planning time. All of these algorithms are known as *backwards algorithms* since they explore the state space around a single goal outwards and towards the starting state rather than the opposite approach. Lifelong Planning A^* (Koenig et al. 2004) is an alternative incremental planner, but is presented as a forwards algorithm. In 2006, (Ferguson & Stentz 2006) introduced the Field D^* algorithm for cell-based data structures only, which allowed plans to interpolate the costs along the edge of a cell, reducing the discretisation effects of the data-structure, and helping to further reduce the plan costs.

For agent-based planning systems that have non-holonomic properties (where the agent's state depends on its prior states, or equivalently, the agent is subject to differential constraints), the 1990s saw many applications use the continuous methods described in the following section. In the mid-2000s however, a discrete approach known as 'state lattice planning' was developed by Pivtoraiko & Kelly (2005*a*,*b*), and extended in Knepper & Kelly (2006), Howard et al. (2008). The approach builds a set of *path primitives* by passing achievable control inputs through a forward kinematic vehicle model. The set of primitives is pruned to just those whose end points lie at discrete states in a regular grid, which is then known as the *state lattice* (Kelly & Nagy 2003). When assessing the cost of a plan, some function of the data in the region traversed by the primitive is accumulated, which means that although state lattice approaches are discrete algorithms, they can operate over both discrete and continuous data structures.

There are several advantages of this technique, including that the completeness guarantees of discrete algorithms are still available, and optimum plans (with respect to the path primitives in the lattice) are achievable. In addition, plans computed in this manner are intrinsically feasible for immediate execution by the agent and may not need any post-plan smoothing steps, since they are built out of agent-specific path primitives.

2.4.2 Continuous Algorithms

Continuous algorithms operate in a continuous domain, and find a plan either by following a set procedure, or by sampling states in this domain and then planning through them using a discrete algorithm. In the former category are methods such as potential field

techniques, which treat the continuous space as an attractive and repulsive force field, and guide the agent in the direction of its force vector at any point. This representation is commonly used to represent obstacles and goals as the repulsive and attractive regions, respectively (Borenstein & Koren 1991, Kavraki & LaValle 2008). Other methods in this category include gradient descent approaches which treat goals as potential wells and follow the steepest descent vector at any point. The Vector Field Histogram (Borenstein & Koren 1991) uses a discrete grid as a data structure, but then uses a procedure which produces continuous steering commands, rather than a sequence of discrete actions. All these methods are vulnerable to getting stuck in local minima however, and require additional modifications to find valid solutions when they are present (Kavraki & LaValle 2008).

The second category is the class of *sampling algorithms*, and is typically used for high dimensional spaces over which it is impractical to perform an operation on all states. LaValle (2006, chap. 5) describes a general template, which is depicted in Algorithm 2.2. In this procedure, $\mathcal{G}(V, E)$, is an undirected graph which is built up by the algorithm. Upon successful completion of the procedure, $\mathcal{G}$ encodes one or more feasible solution plans. X_I is the set of initial vertices inserted into the graph, and typically contains x_I, x_G, or both, depending on the particular sampling algorithm being used. Particular sampling algorithms can be instantiated by specifying the type of Local Planning Method (LPM).

Algorithm 2.2 A general template for sampling algorithms (after LaValle (2006, chap. 5)).

```
1: Instantiate G(V, E), an undirected graph
2: Insert X_I into V
3: while G does not encode a solution plan do
4:    Choose a vertex q_cur ∈ V for expansion
5:    Invoke the Local Planning Method (LPM)
6:    Check vertices returned and add any within C_free into V
7:    Check edges returned and add any wholly within C_free into E
8:    if G encodes a solution plan then
9:       return SUCCESS
10:   else if Termination condition is met then
11:      return FAILURE
12:   end if
13: end while
```

The chief benefit is that the space of obstacle regions, $\mathcal{C}_{obs}$, and free space, $\mathcal{C}_{free}$, do not need to be known fully. The obstacle space is instead represented implicitly, because the algorithm is provided with a collision checking function capable of determining whether a

given state is inside or outside the obstacle subspace. The sampling algorithm need not know the details of this collision check or the definition of the obstacle subspace, yet can still find valid solutions if the sampling strategy is effective.

To be effective, the sampling strategy must lead to dense coverage of the state space. If the strategy achieves this as the number of samples tends to infinity, the algorithm is known as *resolution complete*. If the strategy involves random sampling and it achieves dense coverage with probability $p = 1$, then the algorithm is *probabilistically complete*. This implies that as the number of samples tends to infinity, the probability of the algorithm finding a solution also approaches one.

The two most prevalent sampling algorithms were both developed in the late 1990s. They are the Rapidly-Exploring Random Tree (RRT) (LaValle & Kuffner 1999) and the Probabilistic Roadmap (PRM) (Kavraki et al. 1996), however there are many variations and incremental improvements to both these techniques (Hsu et al. 2002, Kavraki & LaValle 2008). An RRT operates by sampling points in the state space using a Voronoi bias, meaning that a random sample is drawn from inside the largest Voronoi region (Dirichlet 1850, Voronoi 1907) at each iteration. The sample is checked to ensure it is not within the obstacle subspace, and the LPM finds a short obstacle-free plan segment to connect it to the nearest other such segment (splitting an edge in two if any point along it is closer to the sample than either end vertex). Thus, the sampling strategy naturally builds a tree of plan segments that are obstacle-free, and which rapidly cover the entire extent of the state space.

It is often sufficient to sometimes select the goal instead of a random sample (with a low probability; LaValle (2006) suggests $p = 0.01$), and to stop sampling once the goal is connected to the tree. By noting the parent edges or nodes of new edges whenever they are created, it is possible to obtain the plan from start to goal by simple backtracking. Alternatively, the tree structure can be used as the input for most of the discrete graph search algorithms described in the previous section. Although in some domains it is possible to incrementally adjust the state space in such a manner that the tree can be adjusted simultaneously, RRTs are typically used for a single plan, and reconstructed as necessary for use in replanning scenarios.

A PRM operates similarly, but is intended for use in domains where multiple plans are performed over a static state space. As a result, the PRM expends more effort in connecting each new sample to all the existing samples in a local region, using a collision checking

function embodied in a *local planner*. Where an RRT connects a new sample to just the nearest branch of the tree, the LPM used by a PRM connects it to all nearby samples, as determined by some neighbourhood function (such as the nearest n points, all points within a radius r, or all points visible from the sample). Having created this *roadmap*, queries can be performed for many start and goal pairs, or the entire *all-pairs* solution can be found with the Floyd-Warshall algorithm (Cormen et al. 2001, chap. 26) or equivalent.

Unlike RRTs, PRMs form directed graphs which means that as the number of samples added to the map is increased, the solution cost of a given query tends towards the global optimum. For a standard RRT, additional samples add further branches to the tree, but will never result in a lower cost plan from the start to the goal since the tree forms a graph where every vertex has an in-degree of one, meaning there are no alternative routes to a given vertex. Balanced bi-directional RRTs (Kuffner & LaValle 2005) and formulations with multiple trees (Bekris et al. 2003, Strandberg 2004) are also able to reduce solution costs as the number of samples increases (in general), and often add further parameters that affect the balance of computational time and solution cost.

A recent algorithm that combines the sampling approach of RRTs and PRMs with the heuristic guidance of graph search algorithms such as A^* is described by Likhachev & Stentz (2008). This algorithm, known as R^*, can be thought of as an RRT in which a valid connection between two nodes is the solution to an A^* search. The key idea is to abandon these shorter A^* searches if they appear to require a long time or large amount of memory. This has the benefit of relying less on the heuristic function and thus avoiding the expansion of numerous nodes inside local minima, however the downside is that optimality guarantees are lost, and replaced by probabilistic bounds on the solution cost.

2.4.3 Anytime Algorithms

Anytime algorithms are a class of algorithms which iteratively recompute an optimisation with decreasing upper bounds on the cost of the solution (Zilberstein & Russell 1995). In the case of heuristic graph search anytime planning algorithms, the key operating principle derives from the fact that inflating the heuristic by a scalar value $\varepsilon \geq 1$ (and rendering it inadmissible) results in an upper bound on the solution's cost of ε times the optimal cost (Pearl 1984), and typically reduces the planning time significantly (Pohl 1970, Thayer

& Ruml 2008). For sampling-based planners, a similar effect can be achieved by varying the density of the sampling strategy, as described above, or by using an anytime heuristic algorithm to search the sampled graph (Belghith et al. 2006). For other systems it is often possible to vary the resolution of the underlying data structure, or adjust system parameters which affect the accuracy of the plans. The purpose of each of these strategies is to allow tuning of the trade-off between the accuracy of the solutions, and the computational time required to produce them. Anytime algorithms are particularly significant for this thesis because they are a precursor to the full TOPE process, in that they are able to automatically adjust the weighting factor such that the total planning and execution time is typically reduced.

In domains where the execution of a sub-optimal plan with this upper bound is acceptable, one can design a system which continues improving the solution whilst making use of the previous iteration. Algorithms that perform a succession of Weighted-A^* searches with decreasing weights or the Anytime Heuristic Search algorithm (Hansen & Zhou 2007) operate in this manner, but have the significant deficiency that they are not incremental, and all prior information is discarded when deflating the heuristic and beginning a subsequent iteration. The Anytime Repairing A^* algorithm (Likhachev et al. 2003) corrects for this by retaining the algorithm's state across iterations, and efficiently reordering the internal data store to allow subsequent plans to perform the minimum computation required.

Further improvement is provided by the Anytime Dynamic A^* algorithm (Likhachev et al. 2005*a*,*b*) (also known as Anytime D^*), which is a backwards incremental form of the previous algorithm that can account for changes to the weights of edges in the graph. The state associated with each node is extended, by labelling each node either consistent or inconsistent, referring to whether or not the cost-to-come function is correctly propagated to this node. The algorithm's internal data store is then sorted by a function which accounts for these labels. This algorithm can also account for the starting point of a plan changing in any iteration, since the backwards incremental structure focusses the search around the goal (LaValle 2006). This feature is particularly useful for scenarios such as route finding for autonomous agents, whose movements continually alter the starting point of subsequent plans.

Anytime algorithms can be constructed in two forms, with subtly different results. Contract algorithms are run for a specified length of planning time and return the best solution found

in this time (i.e. they are given a contract to return a plan in a given time) (Russell & Zilberstein 1991). Interruptible algorithms are run until the optimum solution is found, but can be interrupted at any point prior to this, returning the best solution found thus far (Dean & Boddy 1988, Boddy & Dean 1989). Contract algorithms are typically easier to construct, but there exists a method to convert any contract algorithm into an interruptible form, by running it repeatedly with increased contract lengths each time (Zilberstein 1996, Hansen & Zilberstein 2001). At least in principle, each of the heuristic graph search anytime algorithms described above can be constructed in an interruptible form.

2.5 Replanning Systems

This thesis distinguishes planning *algorithms*; specific procedures to determine a sequence of actions that achieves a goal, from planning *systems*; techniques which make use of planning algorithms to determine a plan of action, which is then executed by some agent. In this section, planning systems that operate in dynamic state spaces are presented, and subsequently labelled *replanning systems*. To motivate this class of systems, the features of a dynamic state space are first described, showing that such spaces are common in interesting problem domains. Next, the two main classes of replanning system suited to dynamic state spaces are explored, again in their historical context. It is shown that hybrid techniques are able to obtain the benefits of both classes without significant overhead. Finally, a collection of replanning caveats is presented, and the ways in which the systems in this section avoid these problems are assessed.

2.5.1 Planning in Dynamic State Spaces

A dynamic state space is one in which the elements of any state vector can change with time (LaValle 2006). Such spaces can be discrete or continuous, and can be stored using any of the data structures described in 2.3.1. Importantly, even static environments can yield dynamic state spaces if they are incrementally discovered. As examples: a UGV with proprioceptive and exteroceptive sensors builds up a map of the environment as it travels; a team of co-operating Unmanned Aerial Vehicles (UAVs) incorporates local and remote exteroceptive sensor information to refine a probability distribution on the location of a target; and the units in a real-time strategy game reveal the terrain behind a “fog of war”

as they traverse it. In examples such as these, the underlying environment is static, but the state space is built up incrementally and is effectively dynamic.

Planning in dynamic state spaces requires a largely different approach to static state spaces, since a single plan is typically unsuitable for an entire scenario. Even a plan which is optimal with respect to the state space at one point in time can be suboptimal at a later point, and depending on the scenario, could even be suboptimal by the time its computation is finished. Thus, dynamic planning, also known as *replanning*, requires the concept of responsiveness. If a plan cannot be computed sufficiently quickly, it can be virtually useless. An important ramification of this is that it is often better to be suboptimal quickly, rather than optimal slowly. In other words, plans which are optimal with respect to the state space are not always optimal with respect to system performance.

There are many rigid solutions to this dilemma, which pick an operating point along the spectrum between slow but optimal, and fast but suboptimal performance. Section 2.5.2 describes deliberative and reactive techniques which mark the two ends of this spectrum, respectively. Section 2.5.3 describes hybrids of both methods which lie somewhere in the middle of the spectrum, and the most prevalent class of hybrids, known as hierarchical planning systems. The most useful of these are not rigid but flexible, allowing their position on this spectrum to be adjusted before starting a particular planning scenario. Figure 2.5 illustrates this spectrum, showing the typical effect upon the cost of execution and the time to react to changed information in the state space, for any choice of operating point on the spectrum[3].

Another significant difference when replanning in dynamic state spaces is that it becomes possible to plan and execute simultaneously. Section 2.5.4 describes the difficulties this induces, and presents solutions to them. These solutions avoid the inherent inefficiencies of the 'Sense, Plan, Act' paradigm which was prevalent in the 1980s and 1990s (Brooks 1985, Zelinsky 1992). Under this paradigm, agents stop executing when the state space changes, and must wait until a corrected plan is available. The purported benefit of this technique is that it guarantees safety, however it is shown in Section 2.5.4 that there are alternatives

[3]Although this diagram is representative of many replanning systems, it is impossible to produce a quantitative analysis of these effects, since they are influenced by the dynamics of the state space itself as well as many other unpredictable factors. The shape of the curves is intended to reflect that it takes an increasingly large amount of effort (greater than a linear increase) to gain the final performance improvements when planning in many problem domains, and that the time to react increases at a greater than linear rate equivalently.

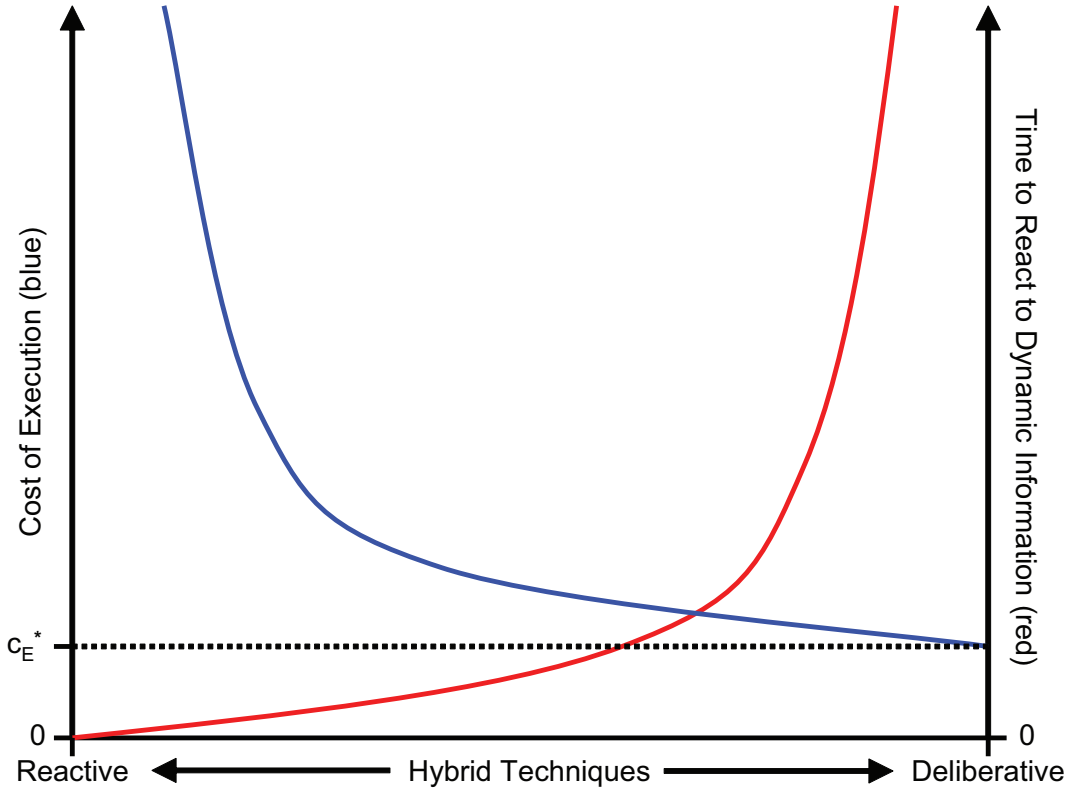


Figure 2.5: Illustration of the spectrum from reactive to deliberative replanning techniques, showing the effect upon the cost of execution and the time to react to changed information in the state space, for any choice of operating point on the spectrum. c_E^* is the minimum cost of execution, such as would be found by a formally optimal planning algorithm.

which can also do so, and which do not require the agent to continually cease execution. Figure 2.6 illustrates that neither ceasing execution while planning nor continuing along a prior plan are always effective, but in an unpredictable state space there is no means by which the best choice can be known. However, Section 2.5.4 further shows that when the frequency with which state space information changes may be arbitrarily high, ceasing execution while waiting for plans becomes untenable.

2.5.2 Deliberative and Reactive Techniques

In and around the 1990s there were two main techniques for replanning in dynamic state spaces: *deliberative* planners, which recompute the entire plan as fast as possible (or when the state space changes); and *reactive* planners, which react to changes in the state space

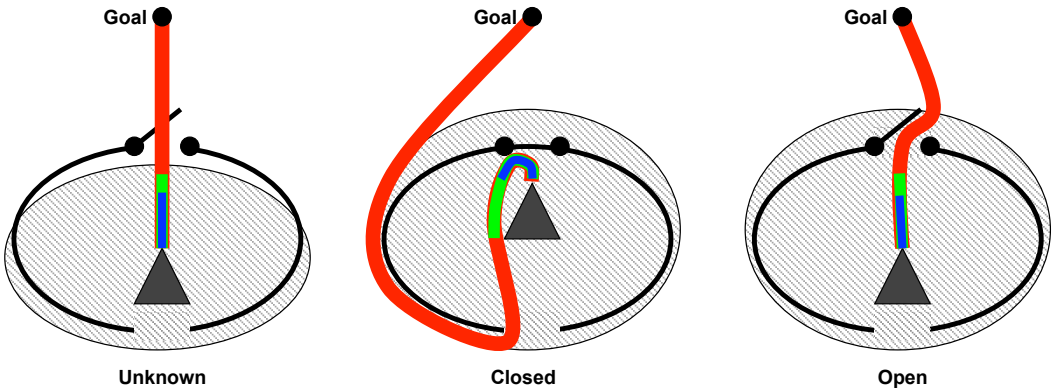


Figure 2.6: Illustration of the choice between waiting for a new plan or continuing along a prior, while a new plan is being computed. In the left image, the agent is guided by the global plan (shown in red) into a cul-de-sac, because the observed region (shaded in grey) does not yet include the gate at the end. Once this gate is perceived and a global plan repair commences, there are two possible choices while it is computed; either the agent follows the committed path (shown in blue) or it remains stationary. If the gate is closed as in the centre image, then following the committed path increases the total execution cost because the agent must now backtrack from its new position. Alternatively, if the gate is open as in the right image, then remaining stationary increases the overall execution time because the changes to the global plan were minimal and the agent could have continued.

by modifying the existing plan. In the deliberative case, all available information from the past and present is used to determine a complete plan to achieve a goal. In the reactive case, only the latest information is considered, typically via a set of condition-action rules to produce a one-step plan (LaValle 2006).

In the deliberative replanning category, Zelinsky (1992), Stentz (1994), and others describe a brute-force replanning approach, where a complete plan to the goal is replanned every time the perceived environment changes. The planning time required for such an approach scales poorly with the distance to the goal, typically $O(n^2)$ for a search optimisation algorithm such as A^* (Hart et al. 1968). However, brute-force approaches can often be made faster by using more efficient data representations for cost, such as quad-trees (Yahja, Stentz, Singh & Brumitt 1998). Anytime algorithms can also speed up the planning (at the cost of optimality) when run repeatedly in a brute-force manner. This has the advantage that a plan can be aborted if the cost information changes significantly, and a new feasible plan made available rapidly (Zilberstein & Russell 1995, Likhachev et al. 2003, 2005*a*).

Incremental algorithms such as D^* (Stentz 1993, 1994) further improve the deliberative

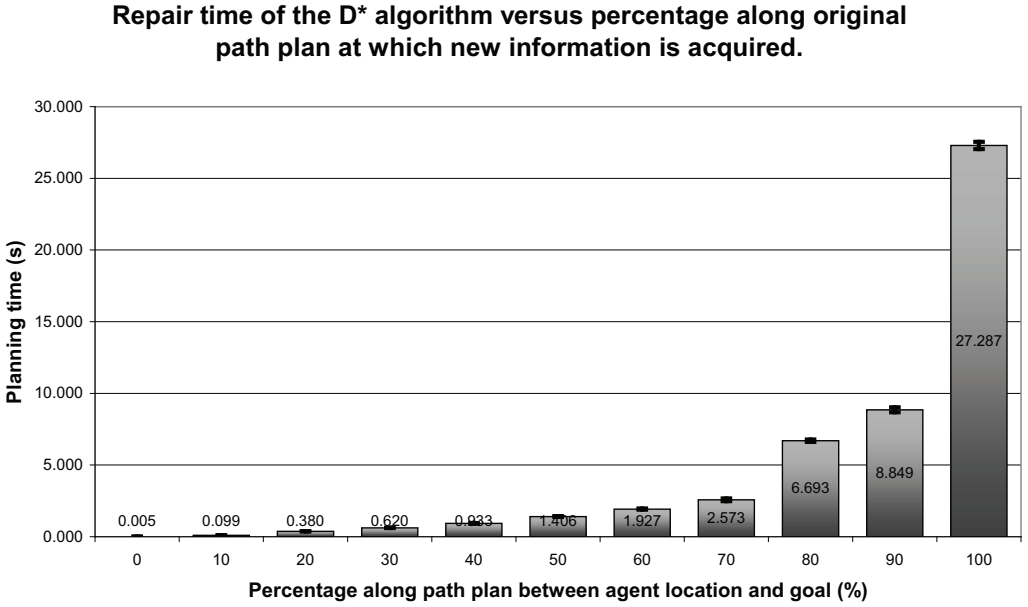


Figure 2.7: Results of an experiment measuring the time taken for the D^* algorithm to repair a pre-existing initial plan, given a fixed size set of changed states at various distances along the original plan, with the error bars showing the 95% confidence interval after ten repetitions. The repair time exhibits a significant increase as the changed information is acquired closer to the goal state. For comparison, the time taken by the A^* algorithm for a full plan is $19.6s$, showing that it is possible for backwards incremental algorithms to exhibit worse performance than their non-incremental equivalents.

performance of a replanning system, because subsequent iterations are typically far faster to compute than a full replan. The one caveat of this class of algorithm is that their benefits are lost if the state space changes near this goal, as is shown in Figure 2.7, which illustrates the results of an experiment designed to verify this for the D^* algorithm, and shows the increase in planning time. The reason for this increase in planning time is that algorithms such as D^* are known as *backwards incremental*, meaning that they expand the search graph outwards from the goal state. As a result, the closer to the goal that changed states occur, the more likely that possible solution plans pass through the changed regions, and thus the more states that must be re-evaluated before the optimum plan can be determined.

In the reactive category, work in the 1980s and early 1990s focussed on *behavioural primitives*, which attempted to tie actions directly to sensor conditions (Brooks 1985, Arkin 1989). In many cases, the output of several condition-action rules could be combined to

yield a more complex response to multiple stimuli. The main benefit of this approach was a virtually instantaneous response to changes in the state space, with the major downside being that the summed response was often not even locally optimal (Murphy 2000). Some reactive techniques incorporated heuristics such as 'heading towards the goal is typically beneficial' (Lumelsky & Stepanov 1986) and 'traversing previously explored areas is typically detrimental' (Korf 1987, Pirzadeh & Snyder 1990). These approaches typically achieved closer to globally optimal performance, but could not guarantee this, and could have detrimental effects in scenarios where the heuristics were inaccurate (LaValle 2006). Later efforts at reactive replanning involved optimal planning over a local subset of the state space in which sufficiently reactive responses were achievable, combined with some form of guidance to bring this local region towards the goal (Allen et al. 2007).

In general, deliberative techniques employ a brute-force approach to plan as fast as possible and buffer state space changes between iterations, whereas reactive techniques are data-driven and respond rapidly to the changing state space. The data-driven approach is prone to problems when the rate of changes exceeds the reactive capabilities of the planning system, and the brute-force approach requires an upper limit on planning time after which an alternative strategy or restart may be needed. The next section describes the vast field of work which attempts to obtain the benefits of both methods whilst avoiding these problems, by way of hierarchical techniques.

2.5.3 Hierarchical Techniques

Much work has focussed on attempting to fuse the speed of reactive and the optimality of deliberative methods into a hybrid system. This is often achieved using a hierarchical system consisting of a reactive lowest level planner, a deliberative highest level, and potentially others in between. Examples of such systems are found in many different planning scenarios, including the RoboCup soccer competitions (Jensen & Veloso 1998), unmanned vehicle navigation (Stentz & Hebert 1995, Kluge & Morgenthaler 2004, Ferguson & Stentz 2006, Allen et al. 2007), and interplanetary exploration (Singh et al. 2000, Tompkins et al. 2004, Carsten et al. 2007). This structure is so prevalent that its terminology is used throughout this thesis. The *lowest level* (or 'lowest layer') planner in a hierarchy is that which is most reactive, and the *highest level* (or 'highest layer') planner is that which is closest to deliberative over the largest subset of the state space. In common with many systems in

the literature, for two-level hierarchies the lowest level is known as the *local planner*, and the highest level as the *global planner*.

Hybrid architectures for planning have existed for over two decades, and illustrate a wide variety of interactions between their components. Many authors cite Brooks' (1985) subsumption architecture as one of the earliest such methods, which allowed each level to subsume the output of lower ones and influence the higher level responses. This architecture was not specifically designed for planning, rather for integrating all aspects relevant to controlling a mobile agent, however the central idea – that higher levels should be able to read outputs from and write inputs to lower levels through a low-bandwidth interface – is present in many other systems in the literature. Brooks briefly notes that each level could be implemented on a separate processor given such an interface, but does not apply this directly to planning.

Blythe & Scott-Reilly (1993) describe integrating reactive and deliberative planning for a household robot, using components known as Hap and PRODIGY respectively. The most interesting aspect of this hybrid system is that while Hap has a list of possible condition-action rules like many reactive planners, invoking PRODIGY to do a new deliberative plan is simply one of these actions, conditioned on a low demand for any other reactive need. The scope of the deliberative plans is limited by the reactive level's estimate of how much time is available before another reactive need takes precedence.

Jensen & Veloso (1998) describe the interleaving of deliberative and reactive planning for dynamic multi-agent domains in the context of the RoboCup soccer challenge. Their approach requires that the state space of the deliberative planner remain constant while planning (or else it is recomputed accounting for these changes), and tries to meet this assumption by discretising this space as a function of the average planning time required. This does not guarantee that the resulting plan is directly useable, but with a suitable discretisation function, a low probability of needing to recompute can be obtained.

Kramer & Scheutz (2003) describe the GLUE framework, and later Scheutz & Kramer (2006) describe the RADIC framework, both of which provide an intermediate interface between otherwise independent reactive and deliberative components. In principle this allows any reactive component to integrate with any deliberative one by writing a specific function to convert between their respective outputs and inputs, however in many hybrid systems such a solution is unnecessary. For example, any system using Brooks' (1985)

subsumption architecture has no need of a translation between levels. In addition, both these frameworks are merely examples of preexisting design patterns[4] and thus the same results can be achieved by a sound design without explicitly using these frameworks.

Gancet & Lacroix (2004) and Joyeux et al. (2007) also employ hybrid architectures combining reactive and deliberative approaches, for single and multi-robot systems respectively. The most interesting element in these strategies is described in Peynot & Lacroix (2006) in which different strategies are employed for various environments and situations. These strategies are described as 'navigation modes', encompassing any or all of perception, decision, and action processes, or parameters affecting them. Information about the performance of the system is obtained from 'context data', such as assessment of the suitability of the terrain for high speed driving, and 'on-line monitoring' of the efficiency of the current navigation mode. A hidden Markov model is used to estimate and switch to the most suitable navigation mode given these sources of information.

Kluge & Morgenthaler (2004) describe a part of the Perception for Off-Road Navigation (PerceptOR) program known as the Raptor system, which fuses three different planning methods into a 'command-arbitration' architecture. A reactive trajectory generator produces steering arcs in response to obstacle proximity, elevation, and slope changes. Next an RRT planner uses local terrain data to continuously find plans that meet the intermediate waypoints produced by a global planner which uses the D^* algorithm. An arbitration step performs a weighted sum of the steering commands produced by the two lowest levels, using the argument that each level is both reactive and suboptimal, and that this sum represents the best combination of both approaches. The global planner has a bounded state space to ensure its plans have an upper limit on computation time.

Kelly et al. (2006) describe the full culmination of the PerceptOR program, which builds on the Raptor system. The planning aspects of this program are hierarchical, and also use the the Field D^* algorithm (Ferguson & Stentz 2006) for global planning in much the same manner. Only one lower level is present; a local planner known as the Ranger which uses a forward-prediction vehicle model to predict the agent's trajectory along a discrete set of velocity and steering commands in the space of control inputs (Knepper & Kelly 2006). Possible trajectories are eliminated if they would cause the vehicle to collide with

[4] In a software context for example, Gamma et al. (1998) describe this as an *Adapter*, and the pattern was well known before it was described in their taxonomy.

an obstacle, and the remaining sets are matched to the closest possible starting cells in a grid representation, with a penalty for slight misalignments. The global planner finds the remaining plan from each starting point with its associated cost, and the overall plan is then the lowest cost combination of local and global plan segments. The hierarchical strategy employed in this work has also been applied to other scenarios, including mobile manipulation tasks in indoor environments (Knepper et al. 2010).

The PerceptOR approach produces a set of feasible rather than optimal local plans, yet the system still achieves close to globally optimal performance. This is possible because the global planner computes the optimal combination of one of these local plans with the global plan, making the overall plan near globally optimal.

Although the PerceptOR program performed multi-agent trials, their aerial vehicle was flown directly above the ground vehicle (Kelly et al. 2006). Such a configuration ensures that the aerial vehicle's information is acquired close to the starting point of each global plan on the ground vehicle, allowing repairs to be computed quickly. As shown in Figure 2.7, the D^* algorithm's performance deteriorates with the distance of new information from the starting point, and hence it is argued that the PerceptOR approach is not suited to data fusion from unconstrained configurations of multiple agents.

The Hierarchical Replanner (HR) presented in Allen et al. (2007) (and reprinted in Appendix A) is similar to the PerceptOR system in that it uses a local and global division, and the D^* algorithm for the global level to speed up any deliberative planning. The HR is designed to be planner and data-structure agnostic, and has been implemented with graph and grid structures, and with Dijkstra's Algorithm, A^*, D^*, and Anytime D^* deliberative level planners. All of these options are configurable at run-time, allowing highly flexible usage. The main limitation of the HR is that deliberative plans are only recomputed when the lowest level planner fails to find a plan within its time limit, which means remote exteroceptive data is not accounted for unless it causes local problems.

One solution to this problem is parallelisation of the planning process, motivated by the reduced cost and increasing prevalence of multi-core processors (Valve Corporation 2010), and the hypothesis that such an approach will allow for near globally optimal results at speeds comparable to reactive methods. Although many reactive and even some efficient deliberative methods are fast enough to render this unnecessary for single agent applications, in most cases however, these same methods are unsuited to the multi-agent data fusion

scenario. The D^* algorithm for instance is optimised for repairing plans due to changed information acquired close to the agent's current location, and shows a deterioration in performance when multiple agents can acquire information anywhere in the planning state space. A parallelised hierarchical planning system is shown to be better suited to this problem, as the different levels of planners account for both local and remote exteroceptive sensor data simultaneously. Furthermore, the hierarchical model is inherently parallelisable, since the separate levels can be computed independently provided that the interface is designed appropriately. Both the HR and PerceptOR can be used in parallelised forms, and the results and benefits of this are further discussed in Section 3.2.

2.5.4 Reactivity and Commitment to Plans

The purpose of this section is to define two key terms that are used extensively in subsequent discussions, and which describe concepts common to all replanning scenarios. The definitions herein are those first described in Allen et al. (2007) and Allen et al. (2009), but similar concepts are found in many replanning systems including Murphy et al. (1997) and Kelly et al. (2006).

The first of these terms is *sufficient reactivity*, and refers to the ability of a replanning system to respond appropriately to changed information in a timely manner. In a dynamic state space, the information known to an agent rapidly becomes *stale*, meaning that it is not known whether the information remains an accurate representation of the state space. Although a particular state space may have a known rate of change, there is no maximum rate that can be specified in general. It is suggested that the only safe way to handle this facet of dynamic state spaces for replanning systems, is to assume a pessimistic view. Under paradigms such as 'Sense, Plan, Act', an agent may be forced to cease execution permanently if the time between information changing is less than the time required to compute a plan in response. As a result, an effective planning system for use in any dynamic state space (rather than just spaces constrained or known to have low frequency dynamics) must allow execution while planning.

It is argued that whether explicitly stated or not, every agent-based planning system features a bound on the length of time for which the agent is permitted to traverse through stale information. In many applications this bound is the time taken to process a batch of

sensory information, whereas in others it is related to the dynamics of the agent. Even in the near instantaneous condition-action rule approach of some reactive systems, there is a finite evaluation time involved during which the agent must still be moving through information known to be stale. For physical agents, even if the scenario dictates that the system should not permit travel through stale data and stops the agent when this would occur, then the agent's inertia will still carry it some finite distance. The limit case occurs for virtual agents with no inertia, yet there must still be a finite processing time between the state space changing and the agent acquiring this information.

Whether or not a planning system accounts for this feature, it is present in any agent-based system, and thus motivates a description of reactivity which accounts for it. The following equation defines the conditions required for a planning system to be sufficiently reactive;

$$max\,(t_P) \leq t_S \tag{2.1}$$

where t_P is the time required to compute a plan, and t_S is the maximum length of time for which the agent is permitted to traverse through stale information.

If Equation 2.1 can be guaranteed to hold, then this is equivalent to saying that a new plan will always be available to account for new or changed information prior to an agent overshooting its bound. As a tangible example, if t_S is equal to the stopping time of a physical agent, then a new plan (which could be just a stop command) will be available before the agent reaches any locations outside its stopping region.

Figure 2.8 illustrates this concept visually, showing three regions of the state space for an agent, and the effect of an obstacle being detected in each. If Equation 2.1 holds, then the region in which the agent must reduce the rate at which it executes actions is eliminated. This helps to simplify the logic required to implement the planning system, and may improve performance if the required changes to keep planning sufficiently reactive are less detrimental than otherwise requiring the agent to reduce its rate of execution for certain obstacles.

The second term to define is *commitment*, referring to the subset of a plan which an agent has started executing, and which cannot be revoked. This concept is needed to avoid detrimental backtracking effects when switching to a new plan from the one being executed. Figure 2.9 illustrates a typical replanning process in a dynamic state space, where planning

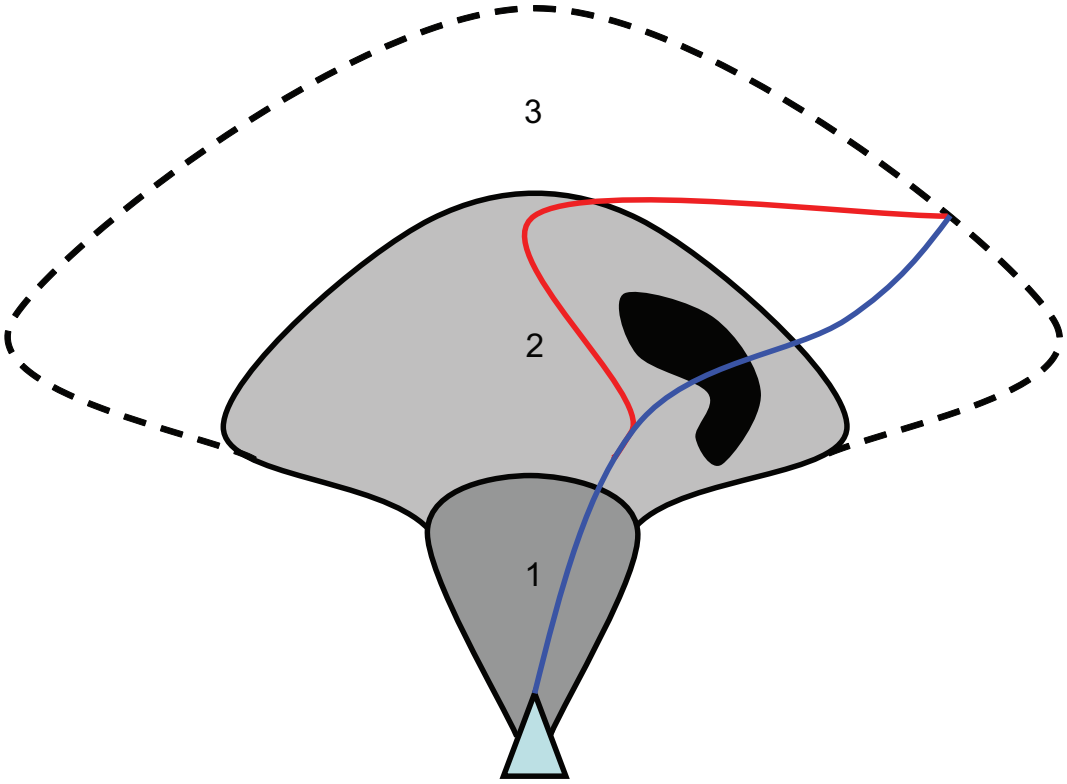


Figure 2.8: Regions of the state space classified by the required actions should an obstacle be detected within them.
Region 1 (dark grey) is the 'region of inevitable collision' (LaValle 2006, Kavraki & LaValle 2008), as any obstacle detected on the agent's plan of execution (the blue line) within this region is unavoidable given the agent's dynamics. For a physical agent, the boundary of region 1 reflects the stopping distance given the agent's current velocity.
Region 2 (light grey) is the 'region of necessary slowdown', as it is impossible to compute a plan to avoid an obstacle on the agent's current plan of execution before the agent collides with it. However, if the agent can be made to reduce the rate at which it executes actions (by an amount proportional to the time to collision), an alternative plan can be found (the red line).
Region 3 (white) is the 'region of confidence', as the time to collide with an obstacle on the agent's current plan of execution is greater than the upper bound on planning time. If a plan which avoids this obstacle is possible (given the configuration of the state space) it will be found, or else no plan is possible and the agent can cease execution prior to collision.
If Equation 2.1 holds, then region 2 is eliminated with region 3 growing to take its place. This is because if $max\,(t_P) \leq t_S$ then all obstacles outside of region 1 meet the definition of the 'region of confidence', and can either be avoided with a sufficiently reactive plan, or the agent can be stopped prior to collision.

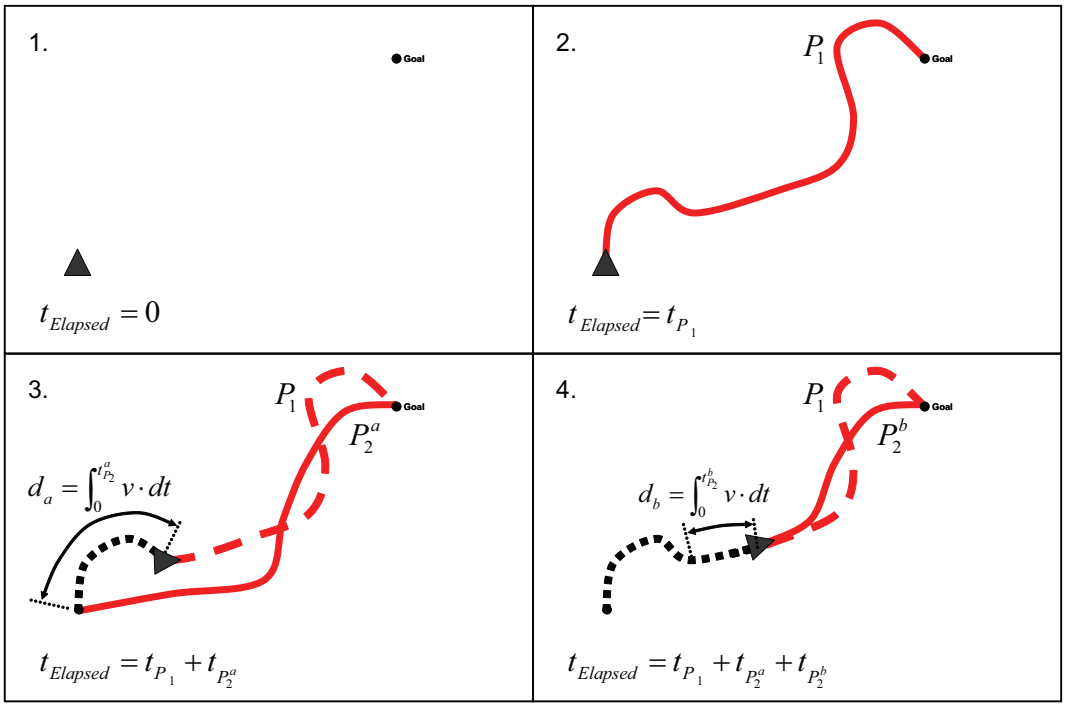


Figure 2.9: Time-series depiction of a typical replanning process in a dynamic state space.

and execution are performed simultaneously, but without any sense of a committed subset of the plans. Although the figure is presented as a 2D path planning problem for clarity, the analysis and motivation are valid for other planning applications. In the figure, solid red lines are the agent's current plan, dashed red lines are previous plans, the dotted black line is the path that has been executed by the agent, which is represented by the black triangle.

The first panel shows the initial state of the scenario; no plan has been computed, and the agent is at its initial position. The second panel shows the state after the first plan, P_1 has been computed, taking time t_{P_1}. The agent then starts executing this first plan while the second plan is being computed. Importantly, this second plan starts from the agent's current location. The third panel shows the state after this second plan, P_2^a, is completed, having taken a planning time of $t_{P_2^a}$ during which the agent has travelled a certain distance along the first plan. Due to this concurrent planning and execution, the starting point of the second plan is no longer the agent's current position, and must be 'snapped' to the agent's location, taking time $t_{P_2^b}$. The fourth panel shows the corrected P_2^b plan, and the process repeats the third and fourth panels until the goal is achieved.

If the agent is continually moving, the snapping process (recomputing a plan such that the starting point is coincident with the agent's current location) shown in the third and fourth panels is unworkable, because in the time required to snap the plan, the agent has moved again, and so on ad infinitum. By committing to a subset of each plan irrevocably, and starting subsequent plans from the end of this *committed plan*, the system effectively achieves the same result. The replanning process thus goes directly from the second to the fourth panel with planning time $t_{P_2} = t_{P_2^a} + t_{P_2^b}$.

Ideally, this subset would be exactly that part of the plan executed in time t_{P_2} (the distance $d = d_a + d_b$ in the example), however this cannot be known until after P_2 is computed. Let C_i be the committed subset of the i-th plan in the procedure (P_i), and t_{P_i} and $t_{E_{C_i}}$ be the time taken to compute P_i and execute C_i, respectively. (The subscript i is dropped when these variables are used in an equation that holds for all iterations.) If $t_{E_{C_i}} > t_{P_{i+1}}$ then the reactivity of the system is limited. Alternatively, if $t_{E_{C_i}} < t_{P_{i+1}}$ then the agent must wait once it reaches the end of C_i[5].

The solution is found by considering the concepts of commitment and sufficient reactivity together. The planning time for a sufficiently reactive planning system cannot exceed t_S. Thus, the commitment length can be up to t_S before it impacts upon the reactivity of the system. It should also not be less than the maximum planning time, or else the agent may have to wait at the end of the committed plan. Thus:

$$max\,(t_P) \leq t_{E_C} \leq t_S \tag{2.2}$$

is an inequality that specifies the design constraints when determining the commitment length, t_{E_C}. In any applications where plan performance can be improved if additional time is available, such as when using an anytime algorithm, then all three variables should be equal.

Many works seen in the literature use a simpler approach of estimating the expected planning time, predicting the future pose of the agent after this time, and then planning from this point (for example, the PerceptOR system (Kelly et al. 2006) and many related projects use this method). This has the significant disadvantage that it fails to consider the concept

[5]Note that the committed plan cannot simply be lengthened if the agent reaches the end, as this would require the starting point of the plan being computed to be moved, which reintroduces the 'snapping' problem from Figure 2.9 that the commitment concept is intended to solve.

of sufficient reactivity, and fails to provide any guarantees of agent safety. In addition, if the prediction is not sufficiently accurate, backtracking may still occur.

These concepts can also be applied to hierarchical replanning systems. Either the system should be constructed such that the maximum planning time for any level is less than the stopping time, as suggested by Kelly et al. (2006), or else all levels should initiate plans starting from the end of the commitment length, as suggested by Allen et al. (2009). In the latter case, the system is sufficiently reactive if Equation 2.2 holds for the lowest level, by the same arguments as above. Furthermore, the system achieves closer to globally optimal performance if this equation holds for higher levels in the hierarchy. This follows because, as suggested in Section 2.2, globally optimal performance is achieved when plans are optimal with respect to the entire state space and can be computed instantaneously. Higher levels in the hierarchy produce plans that are optimal with respect to larger subsets of the state space than lower levels, thus the higher the level that achieves sufficient reactivity, the closer to globally optimal performance.

2.5.5 Meta-Reasoning and Utility-Guided Search

Continuing on the theme of ensuring sufficient reactivity while deliberating as much as possible, this section discusses the fields of *meta-reasoning* and *utility-guided search*, which aim to interleave planning and execution in order to find a balance between the search time and execution cost[6]. In brief, meta-reasoning can be considered as reasoning about the amount of resources available with which to reason about a problem, and utility-guided search as a set of techniques that take a utility function and optimise the operation of a search algorithm with respect to this function, rather than cost.

The field of meta-reasoning arose in the late 1980s and early 1990s (Horvitz 1987, Russell et al. 1991) and was distinguished by the concept of reasoning about the nature of solutions to problems, without actually computing them. Given a set of possible approaches and likelihoods on the outcomes, algorithms such as those described in Russell et al. (1991) could select the appropriate approach and use it to compute the solution. Meta-reasoning approaches almost all determine the expected utility of an approach, and the expected

[6]Utility-guided search is also often referred to as online planning, time-aware planning, or real-time search.

resource requirement, then use a pre-determined trade-off between these two measures to select the best approach.

Compared to the TOADM approach, the most significant deficiency of meta-reasoning techniques is their assumption that resources, such as computational time, are finite. By contrast, there is not necessarily any limit to the resources that a TOADM implementation may use, however it will choose the approach that best optimises the utility function, which penalises excessive use of these resources. A further difference is that the TOADM approach accounts for the time required to perform the meta-reasoning itself, as this involves the use of resources from the point of view of the TOADM equation. As will be seen in subsequent chapters, this difference allows the TOADM approach to be used in situations that meta-reasoning does not consider, such as dynamically adjusting the set of approaches over which subsequent meta-reasoning is to be performed[7].

The field of utility-guided search developed as a response to some of the deficiencies of anytime algorithms described in Section 2.4.3, which generally use a fixed policy to reduce the heuristic weighting factor, ε, over time (Ruml & Do 2007). An anytime policy determines the initial value, ε_0, and the technique and amount by which it is deflated in each iteration. The downside of this fixed policy is that different deflation techniques and amounts result in different behaviours, and so a trial and error approach is required to determine appropriate choices in any given scenario. Furthermore, the correct choices will change with the scenario, meaning that a trial and error approach is not viable in dynamic scenarios.

Utility-guided search starts with a specified utility function that provides a trade-off between the time required to search and the cost or reward of executing the solution. Algorithms such as BUGSY by Ruml & Do (2007) or DTOCS by Lemons et al. (2010) modify an existing search algorithm to optimise this utility function, rather than the expected cost or reward. This is the biggest deficiency of utility-guided search algorithms compared to the TOADM approach; they require modification of an existing algorithm, or the use of a specific utility-guided algorithm. By contrast, a TOADM system wraps an extra level of reasoning around any existing algorithm, meaning it can be used in far more situations, and can be easily applied to new and better algorithms when they are made available.

[7]Russell et al. (1991) and others describe this as "meta-meta-reasoning" and refer to "the problem of infinite regress", where computation at one level is required to determine the manner of computation for the next level and so on. The TOADM framework is formulated such that this does not devolve into a series of nested levels of reasoning, but is all handled within the one TOADM equation.

Where utility-guided search techniques achieve success by exploiting the characteristics of particular algorithms, the TOADM approach can obtain similar success while treating the underlying algorithms as black-box functions.

Meta-reasoning and utility-guided search techniques employ different approaches, but aim to solve broadly similar problems. Both techniques require a pre-defined utility function in order to determine the effect of their decisions, and both formulate this function as a weighted sum of the resources or time required for computation, and the cost or reward of the solution. As will be seen in the following section, the TOADM approach also requires such a function, but formulated in a different manner to a weighted sum, in order that certain useful properties can be exploited.

2.6 Performance Assessment

The previous sections of this chapter have described the data structures and algorithms used for replanning systems in dynamic environments, and shown a collection of such systems from the literature. This section describes metric functions which are used to assess the performance of both planning algorithms and planning systems. This distinction is important, and the terms *internal metric* and *system metric* are used to refer to each type of performance assessment, respectively.

An internal metric is a function that takes two states and returns a real scalar value:

$$M : X \times X \to \mathbb{R} \tag{2.3}$$

Typical examples are the Euclidean distance between two states, or the cost accrued by taking an action to transition from one to the other. LaValle (2006) has a thorough definition of the axioms required for a function to be a true internal metric, which include non-negativity and the triangle inequality, among others.

Certain classes of algorithms, notably those which compute optimal plans, require an internal metric in order to evaluate actions while computing the plan. In the case of optimal graph search algorithms such as A^*, D^*, and their variations, an admissible heuristic function of the same dimensionality as the metric is also required. As described earlier, an admissible heuristic is one which is guaranteed to equal or under-estimate the true cost of

the optimal plan between two states. This is necessary because it is used to guide the search towards the goal, by indicating the best possible connection between two states, and only evaluating higher cost connections when all alternatives have been evaluated and found to be worse.

A system metric is a function that takes n variables, v_i, with associated weights, k_i, and returns a real scalar value that quantitatively describes the performance of the system:

$$m = M\left(k_1 \cdot v_1, \ldots, k_i \cdot v_i, \ldots, k_n \cdot v_n\right) \tag{2.4}$$

This definition is sufficient to cover system metrics that simply report the sum of the internal metric values for each action in a given plan, and also systems which assess values that an internal metric has no access to, such as the number of changes of direction required to execute a discrete 2D plan in a 4-connected grid, which depends on the sequence of actions rather than individuals.

While some planning scenarios will have no need of an internal metric, all must have a system metric in place. For example, a scenario with the objective of 'achieve a goal' can use any plan whose end state is a goal, without needing to determine how efficient or effective the plan was, and thus without an internal metric. However, a system metric is still required to specify whether or not a goal was achieved, even though the output is simply success or failure. A system metric is a measure of how effectively the objectives of a scenario have been met. An internal metric assesses performance while the system is in use, and its results can provide a form of feedback that improves the final system metric value.

Analysis of the planning systems and techniques in the literature with respect to the time of their use shows a clear trend in the objectives used for planning scenarios, and the corresponding types of metrics employed. Early planning systems started with simple objectives such as 'achieve a goal', and used system metrics only to determine performance at the completion of the scenario (Pohl 1970, Brooks 1985). As planning techniques developed, the objectives were enhanced to 'achieve a goal sensibly', with the system metrics increasingly being used to assess performance while the scenario was in progress (Rowe 1990, Pirzadeh & Snyder 1990, Blythe & Scott-Reilly 1993, Jensen & Veloso 1998). Later improvements allowed objectives such as 'achieve a goal optimally', and internal metrics began to be used in

addition to system metrics (Stentz & Hebert 1995, Likhachev et al. 2005*a*, Allen et al. 2007, Carsten et al. 2007, Howard et al. 2008). These internal metrics were used to guarantee optimal plans with respect to the available state space information, and planning techniques making use of the dynamic programming principle used these guarantees to improve system performance too.

Section 2.6.1 describes typical metrics that measure the cost of execution, citing examples from the literature referenced in this chapter. This section also shows that for metrics with multiple parameters, the need to also have an admissible heuristic has some effect on the choice of weightings in the metric. Section 2.6.2 presents the class of metrics of greatest interest to this thesis, which allow each variable to be equated with time. These *time-focussed* metrics allow the required planning time to be equated with the performance of the algorithm, allowing the performance of the entire planning system to be assessed.

2.6.1 Cost of Execution Metrics

In nearly all of the planning systems described in Section 2.5, metrics were used to measure a value that reflected the cost of execution. For internal metrics, this allows the use of planners that attempt to minimise this cost, and for external metrics, it allows comparison with alternative systems that use the same metric.

Probably the most common metric used for both internal and system performance assessment is the shortest distance metric used by Rowe (1990), Rezaei et al. (2003), Allen et al. (2007), and many others. Each action in the plan causes a transition between two states, and this transition has an associated distance that the agent must traverse. The shortest distance metric accumulates the distance between each pair of states in the plan, and the planning algorithm finds the shortest distance plan between the start and goal states. This metric is popular for path planning applications where shortest distance paths also tend to be time, energy, or resource efficient. In addition, it is very simple to encode suitable admissible heuristics for the shortest distance metric, such as the Euclidean or Manhattan distance measures for grid-based planning.

In other examples, such as Tompkins et al. (2004), Allen et al. (2009), multiple parameters contribute to the cost of execution. In many cases, these are completely orthogonal parameters, such as measuring the length of a plan as well as the number of sites of interest that are

visited. In this example, the two parameters must be equated with some weighting factor, where this factor specifies the amount of one parameter that can be traded for the other. This can cause issues when an admissible heuristic is also required, since an internal metric function is required to be non-negative (LaValle 2006). As an example, if this weighting factor is specified as "the plan is allowed to take a $100m$ detour to visit one additional site of interest", then it would be intuitive to specify the metric function as $m = d - 100 * n$, where d is the plan length, and n is the number of visited sites. Such a metric cannot have an admissible heuristic associated with it because it would be possible to continue visiting sites within $100m$ of each other to obtain a negative value. A similarly performing non-negative metric for this example, and for which an admissible heuristic is available, could be $m = d + \frac{100}{(n+1)}$. As LaValle (2006) points out, "virtually any space that has arisen in motion planning will be metrizable" with a non-negative function.

The alternative strategy to using a multi-parameter metric both internally and as a system metric, is to use a simpler metric internally, but devote other levels of control to improving the system metric value. Such a technique is employed by Kelly et al. (2006), Carsten et al. (2007), and others, by way of a hierarchical decomposition of control strategies, similar to that suggested by Brooks (1985).

To summarise the common designs of performance assessment techniques in the literature, it is noted that almost all are used to measure the cost of execution. When only a single parameter or a non-negative weighted sum is sufficient, then the internal and system metrics can use the same function. For multi-parameter metrics, complicated functions with weighting factors are required. To resolve the difficulties with these functions, designers often resort to hierarchical levels of control, with each level effectively adding a parameter to the total value. The effect of this is that the contribution of each parameter is accumulated in sequence, rather than simultaneously, simplifying the problem.

2.6.2 Time-Focussed Metrics

Surveying the systems and techniques described in this chapter, it is clear that although time is considered important for agent-based planning systems, the preference has been to employ distance metrics internally. Designers have relied upon the fact that for physical agent-based systems, where planning is typically used to determine trajectories, distance can be treated as an analogue of time.

Time-focussed metrics are useful for planning scenarios in which the objective is to "achieve a goal, aiming for time-optimality". They are a special case of multi-parameter metrics, defined by the output of the metric being an uninflated quantity of time, where uninflated is used to indicate that the measure is 'real-time', and not weighted by any other scalar factor. If the example in the previous section is rephrased to read "the plan is allowed to take a $10s$ detour to visit one additional site of interest" then it is a form of time-focussed metric. Although many metrics like this example can be time-focussed, the most interesting for this thesis are those for which the output is the expected execution time of the plan, as opposed to an inflated value that just happens to have units of time. This expected execution time is denoted $\hat{t_E}$, and is directly comparable with the required plan computation time, t_P, because they are both *real-time* measures.

The main benefit of such metrics is that they provide the ability to trade-off these two measures, to reduce the total time expended in achieving the goal, which is a function of both t_P and t_E, determined by a time-focussed system metric. This is a significant difference compared to all other metrics which can only trade-off at best between t_P, and c_E. Systems without time-focussed internal metrics would need a separate system metric to determine whether a particular distribution of planning time and execution cost was somehow better. Systems employing a time-focussed metric can determine this trivially, since the sum $t_P + t_E$ is just a scalar.

For example, in any system using an anytime algorithm, the typical approach in dynamic scenarios is to simply let the anytime algorithm continue deflating its heuristic until it reaches $\varepsilon = 1$, or significant changes to the environment are detected (Ferguson et al. 2005, Likhachev & Ferguson 2009). With a time-focussed metric, it is possible to determine whether this use of computational time is actually beneficial in terms of achieving the goal earlier. A technique using this concept is further described in Section 3.4.

Due to their ability to compare planning time with execution time, time-focussed metrics are a critical requirement of solutions to the TOPE problem. Similarly, they fulfil this same requirement for solutions to the TOADM problem by allowing comparison of the time required to compute decisions, and the time-performance of a system utilising the results of these decisions. In both cases, the solution processes use a time-focussed metric to enable evaluation of the total expected time to achieve a goal, given the possible adjustments to the system that could affect the components that comprise this total time.

2.7 Conclusion

This chapter has presented a review of literature from the planning domain, with specific emphasis on planning systems for agent-based scenarios in dynamic environments. The two main areas of research effort in the last 40 years of planning research were shown to involve algorithmic efficiencies resulting in reductions in planning time, t_P, and improvements to the domain representations or the use of optimal algorithms to reduce the cost of execution, c_E. The final section presented the concept of a time-focussed metric, which is used in planning systems to map $c_E \rightarrow t_E$.

For TOADM problems, which include the TOPE problem for planning systems, time-focussed metrics are critical in that they can evaluate the effect of adjustable system parameters upon the total time required to achieve a goal. Where the use of internal time-focussed metrics for planning algorithms allows time-optimal *planning* performance, comparison of this internal metric with the required t_P creates a time-focussed system metric, allowing time-optimal *system* performance. The focus of the following chapter is the use of such metrics in planning systems, to determine the types of strategies available to reduce the combined planning and execution time.

Chapter 3

Adjusting Planning and Execution Time

This chapter assesses the wide range of strategies available to replanning systems in order to adjust the trade-off between planning and execution time. It builds on the concept of time-focussed metrics presented in Section 2.6.2, which map the cost of execution, c_E, to the execution time, t_E. Since t_P and t_E have the same units, this enables the minimisation of the sum of both these parameters.

The techniques described in this chapter all affect the trade-off between t_P and t_E. In most cases, the techniques themselves cannot directly affect t_E, but instead affect the output of the planning system, which is a plan with associated expected execution time, $\hat{t_E}$. The true t_E can only be known after the agent has achieved its goal, so the techniques instead adjust the sum of $t_P + \hat{t_E}$. For static state spaces where only one plan is required, this sum is the expected total time required to achieve a goal. For replanning scenarios, this sum is the expected total time remaining at each iteration, and is still the desired quantity to minimise.

Section 3.1 starts by describing the field of 'time-optimal planning', explaining why the body of literature on this subject is unsuited to applications in dynamic state spaces or replanning systems.

Section 3.2 explores the technique of parallelisation, particularly of the levels of a hierarchical replanning system. It is shown that this technique can significantly reduce the total time

required to achieve a goal in dynamic state spaces, while guaranteeing that performance is no worse than the equivalent non-parallel system.

Section 3.3 presents a series of tuning parameters that affect the trade-off between t_P and t_E, and the resulting total time required for a replanning system to achieve a goal. These parameters are assessed in terms of their applicability to various algorithms, data structures, and problem scenarios, and it is shown how these parameters affect the performance of the replanning systems from Chapter 2.

Section 3.4 presents a novel contribution to the field of anytime algorithms, by determining an admissible lower bound on the optimal plan cost while the anytime deflation process is underway. When a time-focussed metric is also employed, this enables a technique which can detect when there is no potential performance improvement from continued anytime deflation. By aborting the planning process at this point, the technique has the potential to save computational resources.

3.1 Time-Optimal Planning

Time-optimal planning involves the computation of a plan that results in the minimum execution time, t_E, but typically ignores the time required to compute this plan, t_P (Heinzinger et al. 1990, Shiller & Dubowsy 1991). There are many situations in which this is an effective technique, including applications such as circuit board component placement and automotive assembly robotics (Bobrow 1988), high-rise elevator scheduling (Schlemmer & Agrawal 2002), unit routing in computer games (Rabin 2003), and space-filling planning algorithms such as the Distance Transform (DT) (Jarvis 1984). In all these examples however, either the state space is static, planning time is either insignificant or considered to have no bearing on performance, or the planning time is constant across all instances.

Time-optimal planning requires at least one of the above conditions to hold, yet this is unattainable in many scenarios. For realistic problems without oversimplifications, planning time is never completely insignificant, as demonstrated by the ongoing work in the optimisation and design of planning algorithms for increasingly complex state spaces. Computational time is never completely free, and only approaches zero cost when plans are used in perpetuity. Even in a suitable scenario, such as circuit board component placement for example, there is still a benefit from reducing the initial planning time, as it can bring

production forward. The last condition, that planning time remains constant in all cases, is the least likely to be valid. The complexity of the state space has a huge impact on the planning time, to the point that many algorithms suffer catastrophic slowdown in canonical cases (see Figure 2.7 for example). Therefore, in dynamic state spaces planning time is never constant, and for static spaces the previous argument that it is beneficial to reduce this initial or offline planning time still holds. The time-optimal planning technique is also not well suited to applications involving dynamic environments, because the need for continual replanning prohibits excessive computation time.

These arguments indicate that if minimising the total time required to achieve a goal is the overall objective, then the time-optimal planning approach of minimising t_E only is unlikely to be superior to a technique that solves the TOPE problem by assessing the contributions of both t_P and t_E to the total time. Indeed, in not one example in the thorough literature surveys of Garvey & Lesser (1994), Fiorini & Shiller (1996), and Smith et al. (2000) are the contributions of both t_P and t_E considered. The most similar approach found in the literature is the topic of optimal scheduling with setup costs, surveyed by Allahverdi et al. (1999, 2008). In this topic, scheduling problems with a significant setup or switching time are assessed, and algorithms developed to determine when the cost of this time is justified by the savings made by the new schedule. One example is "a facility that produces supplies to photocopiers and laser printers. [The author] pointed out that changing production from one toner to another results in large setup times (generally of the order of days)." However, switching to this new toner can have significant monetary savings, and the scheduling problem is to determine whether the loss of copying time is justified by these savings. While accounting for setup time in a scheduling problem is similar to accounting for computational time in a planning problem, the former is not determined online, and thus the setup time is a one-off contribution and does not incorporate the time required to solve the scheduling problem.

At this point, it is clear that the TOPE problem is different to virtually all prior work, including the related fields of scheduling with setup times and time-optimal planning. The TOPE problem does not aim for time-optimal *planning* performance, rather time-optimal *system* performance. It is also apparent that the field of time-optimal planning is poorly named, as it is in fact a form of planning for time-optimal execution. The remainder of this chapter focusses on techniques and tuning parameters for planning systems that when

combined with a time-focussed system metric, affect the total time required to achieve a goal. When employed in the TOPE problem, these same techniques and parameters enable the minimisation of this total time.

3.2 Parallelisation

Parallelisation is a technique which allows multiple tasks to be executed simultaneously, and is beneficial for almost all replanning systems (Grama et al. 2003). The technique works either by allocating each task to an individual processor, or by scheduling short 'time-slices' in which each task is executed up to some maximum time limit before switching to another task. This section briefly discusses the parallelisation of planning algorithms, but focusses on the parallelisation of planning systems, particularly hierarchies.

3.2.1 Parallelisation of Planning Algorithms

The parallelisation of planning algorithms has been described and implemented by many researchers for many problem scenarios and infrastructure architectures, and frequently for graph search algorithms on shared memory, multiple processor architectures (Arjomandi & Corneil 1978, Grama et al. 2003, Rao & Kumar 2005). While the performance increases are significant, for the purposes of this thesis, it is important to determine the best use of available processors and computing power, hence comparison with the parallelisation of planning systems is necessary. This section argues that for planning scenarios which aim to minimise the total time required to achieve a goal, the parallelisation of planning systems is of more use than the parallelisation of planning algorithms.

One reason for this is the existence of a limit to the performance increases due to algorithmic parallelisation, even if the number of parallel computational units is unlimited. This limit is governed by Amdahl's law (Amdahl 1967), shown in Equation 3.1, which states that any sequential part of an algorithm will limit the maximum performance improvement to:

$$s = \frac{1}{1-p} \tag{3.1}$$

where s is the proportional speed-up of the algorithm, and p is the proportion of parallelisable instructions. Gustafson's law (Gustafson 1988), shown in Equation 3.2, states a

similar result when the number of parallel computational units is specified:

$$s = n - p(n - 1) \tag{3.2}$$

where n is the number of parallel computational units. The parallelisation of planning systems may have similar limits (depending on the nature of the hierarchy), but its main benefit is to avoid the potential ramifications of employing all computational resources on just one task.

3.2.2 Parallelisation of Hierarchical Planning Systems

Parallelisation is especially valuable for hierarchical replanning systems because it allows each level in the hierarchy to compute plans or take actions simultaneously, rather than needing a complex scheduling component. For example, the Hap/PRODIGY planning system of Blythe & Scott-Reilly (1993) could be redesigned using parallelisation to run both components together, rather than only scheduling a deliberative plan when the reactive planner was free. The RoboCup soccer system of Jensen & Veloso (1998) could be redesigned to avoid needing to compromise the deliberative results when coarsely discretising the state space, by instead using parallelisation to run both planners simultaneously.

The Hierarchical Replanner (HR) system used on the CORD UGVs and described in Allen et al. (2007) shows a second motivation for the use of parallelisation in replanning systems. Without parallelisation, the HR only performs a new deliberative plan if the reactive planning level fails (either because the local region is blocked by obstacles, or its time-limit elapses before a plan is computed). Although the system is capable of receiving additional data from exteroceptive sensors that observe beyond the local region, and using data fusion to incorporate the information from other agents, this information is not used by the planning algorithms unless the local level fails. This leads to suboptimal behaviour where the system can have knowledge of obstacles blocking the goal region for some time, but does not react to this knowledge until the agent reaches them.

Parallelisation was added to the HR system in Allen et al. (2009) (reprinted in Appendix B), which described the Parallel Hierarchical Replanner (PHR) and allowed the agent to operate more efficiently in multi-agent data fusion scenarios. The system was designed such

that even in the worst case scenario, the PHR could be no worse than the original HR[1], but had the potential to be far superior at other times. Parallelisation of the HR also enabled the use of anytime algorithms for the highest level in the hierarchy, since this level could continue improving the plans while accounting for changing information.

The main benefit of parallelisation in hierarchical planning systems is that it enables the system to be reactive at higher level in the hierarchy, leading to closer to globally optimal performance (for the reasons described in Section 2.5.4). Higher levels may not meet the criteria of Equation 2.1 to be sufficiently reactive, but if plans are optimal with respect to the subset of the state space over which a particular level plans, then the more reactive this level is, the better the performance. It was shown in Allen et al. (2007) that the HR was able to be tuned along a scale between highly reactive at one end, and closer to globally optimal on the other, by adjusting the size of the local planning region. In the PHR, this tuning is unnecessary since the communication of plans between levels in the hierarchy is asynchronous. As a result, the system is sufficiently reactive if Equation 2.1 holds for the lowest level, and achieves closer to globally optimal performance if it holds for higher levels in the hierarchy.

Both the parallelised PerceptOR system of Kelly et al. (2006) and the PHR have been fielded on UGVs, and have been used in multi-agent scenarios. In both cases, performance was improved over prior system architectures, however the metrics used to judge this were different. Kelly et al. (2006) used a combination of several factors including the distance travelled, time taken, average speed of the agent, and number of human interventions required. Allen et al. (2009) measured a combination of the distance travelled and the 'cost' accrued by traversing rougher terrain. Although neither system employed a time-focussed metric, both show that simply by having a metric with which to assess the results empowers designers to make changes that improve this metric. It is thus reasonable to assume that both systems would be capable of reducing the total time required to achieve a goal, if a time-focussed metric was to be employed. This is verified for the PHR system in Chapter 4, where the use of a time-focussed metric allows an offline static analysis of the effect of various parameter choices to be performed, and the information obtained improves the performance of the PHR system in a dynamic scenario beyond that shown in the experiments

[1]This is true because even if the global planner never succeeds in computing a single plan when running in parallel, if the local planner fails, global planning is aborted then restarted accounting for the changed information. This makes the operation equivalent to the HR and performance is also equivalent.

of Allen et al. (2009).

3.2.3 Comparison of Parallelisation Techniques

A simple thought experiment based on the upgrade from the HR to the PHR replanning systems, demonstrates at least one case in which system parallelisation is superior to algorithmic parallelisation. As described in Allen et al. (2009), the worst case performance of the PHR occurs when global planning takes infinite time but does not fail. In this case, the PHR performs exactly as the HR does, and continues locally planning along the previous global plan. This means that the agent is still progressing, albeit less than globally optimally.

The alternative use of the same computational resources would be to employ two processors to parallelise the global planning algorithm. In such a case, planning time would still be infinite, and the additional processor would have zero benefit. This example applies up to the point that un-parallelised global planning takes $n \times t_E$ worth of time, where n is the number of processors available, and t_E is the time required for the agent to reach a goal using only its local planner. Since parallelised planning takes a minimum of $\frac{n \times t_E}{n} = t_E$ (by Gustafson's Law), it gives no benefit in this scenario.

While this discussion cannot conclusively state that the parallelisation of planning systems is always better at reducing the total time required to achieve a goal than the parallelisation of planning algorithms, there are certainly scenarios in which this is the case. The conclusion of this section is that the use of parallelisation for replanning systems provides two main benefits: the ability to perform multiple tasks simultaneously and avoid the use of tuning factors that would otherwise be required to prioritise them; and the means for the system to achieve closer to globally optimal performance, as higher levels are rendered more reactive.

3.3 Tuning Parameters

Having established that the use of a metric to evaluate system performance is valuable in Section 2.6, this section assesses a wide range of tuning parameters and techniques that affect this performance. If this metric is time-focussed, then these parameters enable the

trade-off between planning and execution time, and can potentially reduce the total time required to achieve a goal.

The results of experiments that vary these parameters for TOPE replanning applications are also presented. In many of the experiments, the scenario is that presented in Figures 2.1 and 2.2, under the assumption that the state space is static and the entire data structure containing it is known in advance. Each figure shows the results as a stacked column graph of the planning (shown in blue) and execution (shown in red) times, versus a discrete set of parameter values, and illustrates that there is always at least one optimal choice of the set of parameters to minimise the total time required to achieve a goal. This point must always be true, since the central limit theorem shows that any finite non-empty set must have one or more minima.

It will be shown in Chapter 4 that each of these tuning parameters exists in the space of possible sets of parameters to a planning system. As such, they can be used in the TOPE process to minimise the total time required to achieve a goal, when the scenario is dynamic and requires a replanning system.

The experimental procedure is similar in all cases. The scenario is implemented within a simulator in such a manner that it is replicable, and that the parameters in question can be adjusted before each run. A model of the agent is used to determine the actual execution time for a given plan in the static state space. The simulation is then run in a brute-force manner for each value in the space of available parameter values, with the resulting planning time[2] and execution time stored in a database. A post-processing step then applies the system metric (simply $t_P + t_E$) to each result to determine the best performing parameter set. The purpose of these experiments is to illustrate that a wide variety of tuning parameters are available within commonly used planning systems. As such, the results are indicative only, and are not intended to reflect the optimum parameter sets outside of the experimental scenarios.

[2]The experiments were performed on a standard X86 architecture within the Microsoft Windows operating system. As a result, despite the use of a deterministic simulator which repeatably returns the same plan and execution time for a given scenario and parameter set, the measured planning time is only accurate to within approximately $\pm 8ms$.

3.3.1 State Space Discretisation Parameters

The first tuning parameter is actually a general category of parameters; those that affect the discretisation of the state space used for planning, and are thus relevant to any system that employs a discrete planning algorithm. Since a representation of the state space is invariably stored in a data structure, these parameters are data structure dependent. Despite this, since almost all data structures have such a tuning parameter, the category itself is generally applicable.

Typically, as the discretisation becomes coarser, t_P decreases but t_E increases. If a conservative approach (such as visualised in Figure 2.3 for regular grids) is used when converting the underlying data to a discrete representation, such that the value stored in each region takes the worst metric value of any data within its bounds, then the effect on t_E is amplified. As the discretisation is made finer, the plans are able to make use of lower metric value regions that are concealed by coarser representations, and t_E decreases. Conversely, since the time required by discrete algorithms is related to the number of states in the space and is a function of the number of states in the plan, t_P increases as the region size is reduced, since this results in more states in the data structure and more possible plans to consider. Any other choice of filter applied to the raw data (i.e. other than the conservative approach of taking the maximum cost value) will have a different affect on t_E, but a time-focussed metric allows a consideration of the effect upon t_P and t_E to be assessed regardless.

The following list describes tuning parameters that affect the state space discretisation for the data structures presented in Section 2.3.1.

Occupancy Grids Can adjust the size of each region to make a coarser or finer grid, with the same effects as described above.

2^N-trees Can adjust the maximum number of data points in each region before it is split into sub-regions. This has the effect of limiting the maximum depth of the tree.

kD-trees Similar to 2^N-trees, the number of maximum data points within a division before it is split into further subdivisions acts as a discretisation parameter. In addition, this technique also affects the time required to build the tree initially, and also the time required to traverse to a particular subdivision and to resolve between data points within a division.

Cell Decompositions Voronoi decompositions normally continue decomposition until each data point lies within its own region. Adjusting the maximum number of splits that are allowed is thus a state space discretisation parameter. Other decompositions typically follow a fixed procedure which can either be aborted at a certain point, or the maximum number of iterations can be adjusted, with similar effects.

Graphs Are typically formed by conversion of one of the previous data structures, in which case the discretisation is affected by the original data structure. If raw data is converted to a graph form then a tuning parameter is almost always apparent during this process. As an example, when converting Open Street Map data representing a road network into a graph, one can skip many nodes along a road to reduce the size of the graph (OpenStreetMap Foundation 2010). A parameter such as the minimum allowed distance between nodes is thus a state space discretisation parameter. Section 3.3.2 below also demonstrates that the sampling density of a continuous domain sampling algorithm affects the graph structure in this way.

Figures 3.1 and 3.2 show the effect of varying the cell size of a regular grid representation of the data from the scenario shown in Figures 2.1 and 2.2. The set of cell sizes (the side length of square grid cells) was $\{0.25, 0.375, 0.5, 0.75, 1, 1.5, 2, 3, 4, 6, 8\}$ metres. The results show that in these scenarios, the cell sizes that result in the lowest total time to achieve a goal (given the discrete set of options) are $2m$ and $0.375m$, for the longer and shorter scenarios respectively. Plans which required longer than $15s$ to compute are not shown in the data, since although they may further reduce the expected execution time compared to other options, they do not further reduce the total time. Also note that for cell sizes of $6m$ and $8m$ in the longer scenario, the goal location is considered to be an obstacle region and thus the A^* search fails immediately. The key point that can be drawn from these graphs, is that the cell size that results in the lowest $t_P + t_E$ from this discrete set is not the finest resolution, as it would be if only t_E was being optimised.

3.3.2 Sampling Density

The concept of sampling density is a tuning parameter that applies to systems operating in continuous domains and using a sampling algorithm[3]. As described in Section 2.4.2,

[3]Alternatively, it can be considered as a parameter in any Monte-Carlo simulation that determines a distribution by sampling randomly until sufficient accuracy is achieved, with sampling-based planning algo-

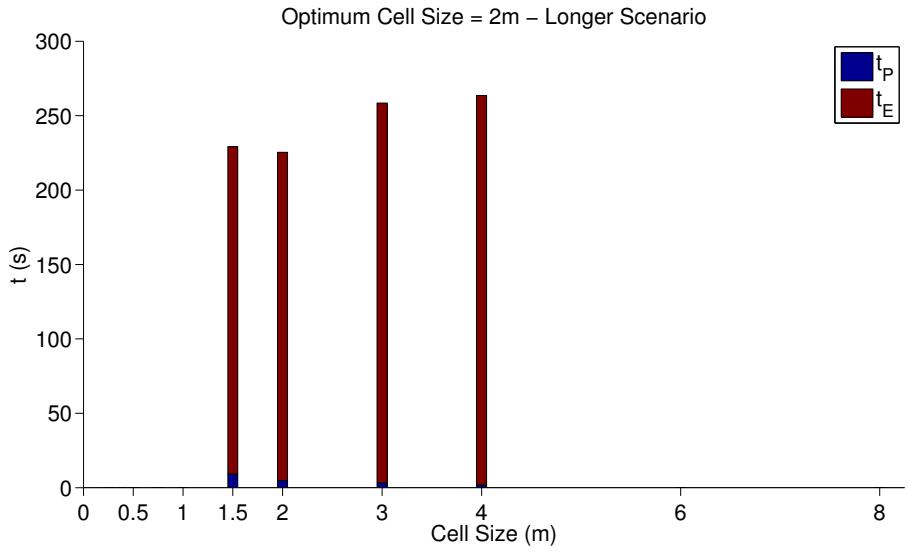


Figure 3.1: Total time required to achieve a goal in the longer static planning scenario versus the cell size used to discretise the grid containing the state space information, using the A^* algorithm. The minimum total time for this scenario and parameter set is $225.4s$, and is achieved when the cell size is set to $2m$.

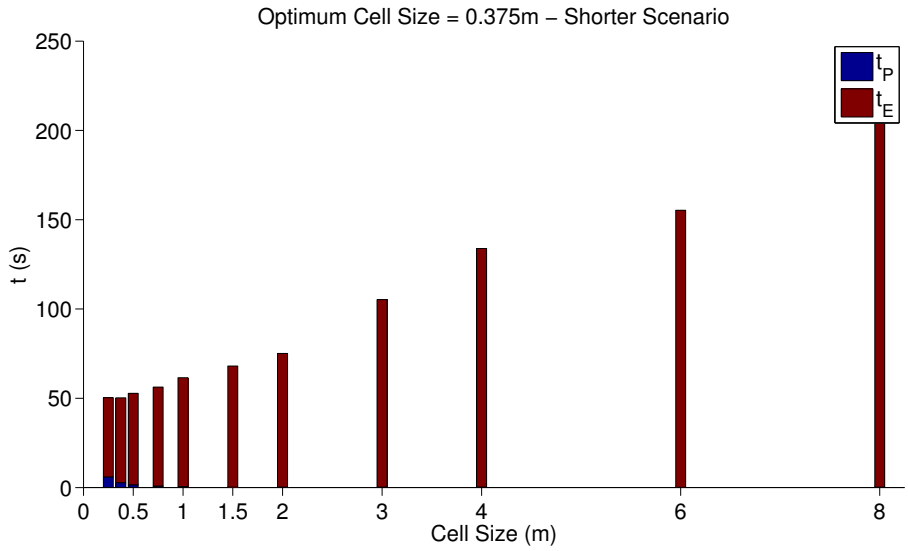


Figure 3.2: Total time required to achieve a goal in the shorter static planning scenario versus the cell size used to discretise the grid containing the state space information, using the A^* algorithm. The minimum total time for this scenario and parameter set is $50.2s$, and is achieved when the cell size is set to $0.375m$.

sampling algorithms build a graph by selecting data points in the non-obstacle regions of the state space and connecting them with feasible plan segments. In the query stage, this graph is then searched with a traditional graph search algorithm, such as A^*. The samples are drawn from a distribution which must be dense in the limit as the number of samples goes to infinity. However, this distribution can be uniform, employ a Voronoi bias, or be biased by any other heuristic function as long as it is dense with probability $p = 1$ in the limit.

The duration (either a time or number of samples limit) for which the sampling stage of the algorithm is permitted to proceed is a tuning parameter that affects the density of the sampled graph. Much like the previous category of parameters, those that increase the sampling density tend to increase planning time but reduce the execution time required to achieve a goal[4].

Figure 3.3 illustrates the effect of sampling density on a PRM implementation. This simplified implementation picks a random sample point within a bounded operating region in each iteration, and connects this point to all neighbouring points within a small radius. Each connection between points has an execution time calculated by finding the minimum velocity at any point along the line between them, as specified by the cost map, and dividing the distance between the points by this velocity limit. The background image in the subfigures represents this cost map as a regular grid, with squares ranging from white to black representing high to low velocity limits, and magenta squares representing untraversable obstacles. The three subfigures show visualisations of the roadmaps (blue) and the plans through these maps with lowest execution time between the start and goal locations (green), for three different sample quantities; 1000, 2350, and 3000.

Figure 3.4 shows the effect of varying the sampling density by limiting the maximum number of samples to this PRM implementation. The maximum number of samples was increased by 50 at a time, from 100 to 3000, and the results show that in this scenario, the optimum limit is 2350 samples[5].

rithms being a subset of this category.

[4] As described in 2.4.2, this is not true for standard RRTs, since they build an acyclic tree rather than a graph. Continuing to add samples after the start and goal are known to be connected by the tree, has no effect on the structure or cost of the optimum plan from start to goal, since further samples merely branch existing edges rather than adding alternatives (LaValle 2006).

[5] Although it may appear from the graph that 2350 samples would be the best parameter value if optimising t_E only, the value of t_E for 2650 samples and greater is slightly lower still.

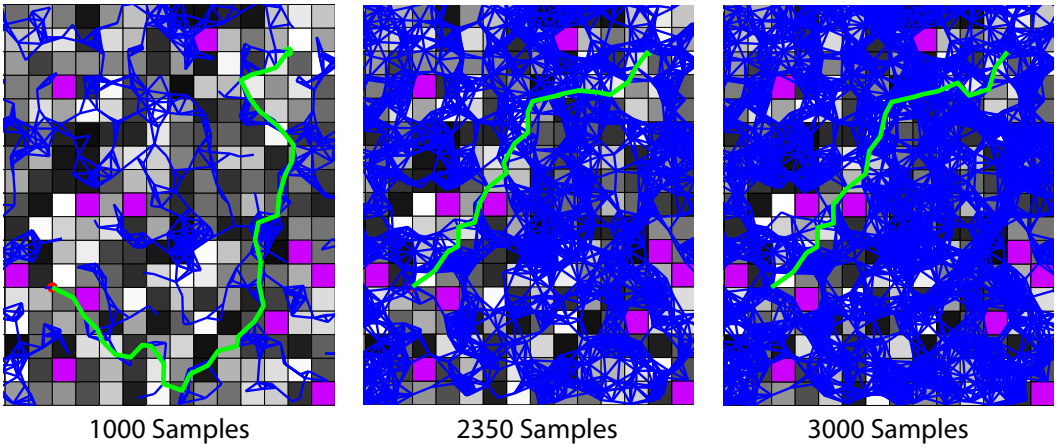


Figure 3.3: Visualisation of the effect of sampling density on the roadmap and optimum plan within this map, for a simplified PRM implementation. When the roadmap is too sparse, the optimum plan within this map has a high execution time, however the time required to compute the roadmap and plan is low. When the roadmap becomes too dense, the optimum plan has a low execution time, but the computation time is far higher.

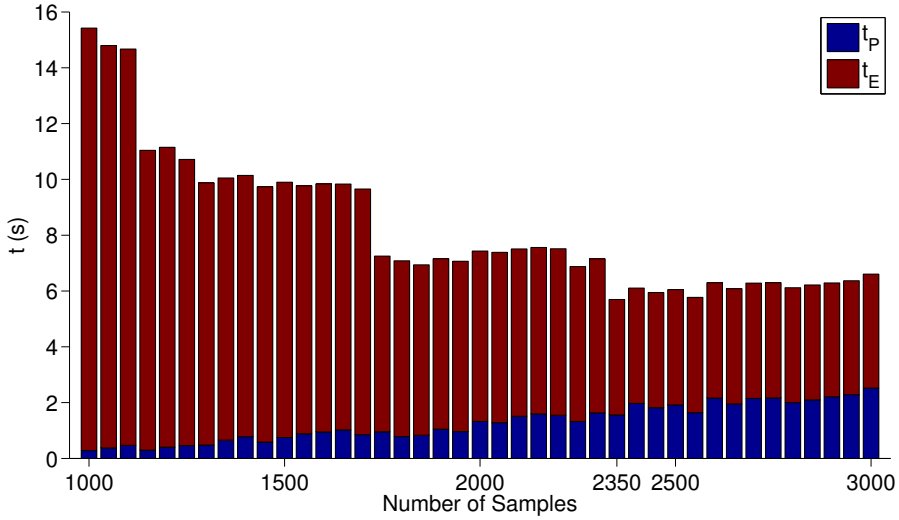


Figure 3.4: Total time required to achieve a goal in a static planning scenario versus the number of samples allowed for a PRM. For this particular scenario, using a uniform sampling distribution and fixed random seed for replicability, the optimum number of samples from the discrete set being assessed is 2350.

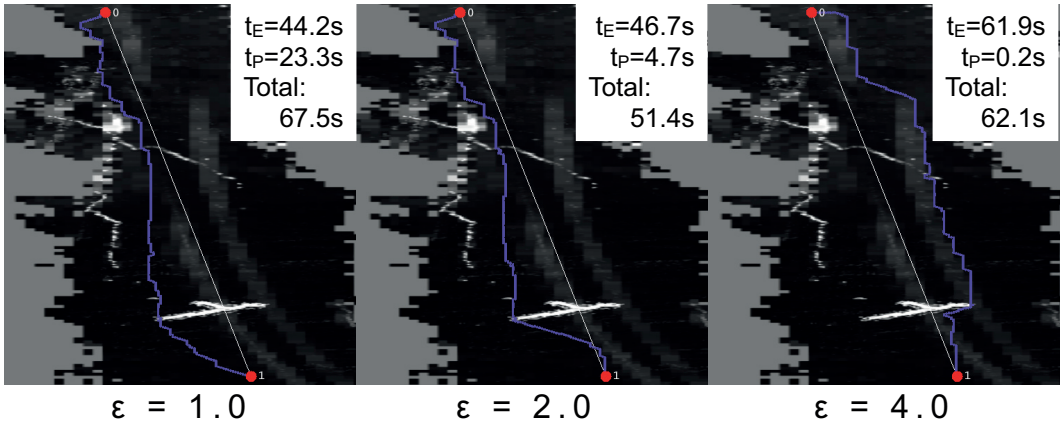


Figure 3.5: Visualisation of the effect of varying the heuristic inflation factor, for the static 2D path planning scenario. For high values of ε, the paths avoid obstacles (as these are unconnected vertices in the graph used by the planning algorithm), and have low planning times, but favour the heuristic approach of aiming directly for the goal too highly. As $\varepsilon \rightarrow 1$, the paths are closer to optimal, but require increasingly longer planning times.

3.3.3 Heuristic Weightings

As described in 2.4.3, multiplying the admissible heuristic of a graph search algorithm by a scalar weighting factor, ε, is able to reduce t_P while guaranteeing that the solution cost is less than the optimum cost multiplied by ε. As a result, ε is a suitable tuning parameter to affect the trade-off between planning and execution time, when the planning algorithm employs a time-focussed metric. It is important to note that decreasing the heuristic weighting factor is by no means guaranteed to reduce the solution cost, merely the upper bound on this cost. It is also common for the minimum cost plans to be attained despite ε being greater than 1, and with all lower ε resulting in the same plan and cost.

Figure 3.5 shows the effect of varying the heuristic inflation factor, by visualising the paths for three different values of ε: 4.0, 2.0, and 1.0, directly on the cost grid for a $0.25m$ cell size.

Figures 3.6 and 3.7 show the effect of varying the heuristic inflation factor, ε, for the scenario shown in Figures 2.1 and 2.2, using the Weighted A^* algorithm and a $1m$ cell size. The heuristic inflation factor values form a set ranging from 1 to 1000 generated by starting from 1000, raising the previous value to the power of 0.95, and including the uninflated heuristic where $\varepsilon = 1$. This discrete set of weighting factors was chosen by drawing on experience

with anytime algorithms, where it provides a larger number of samples as ε tends to 1 and also covers much of the region of the parameter space in which the expected execution time of the paths vary. The results show that in these scenario, the values of ε that result in the lowest total time to achieve a goal (given the discrete set of options) are 1.8167 and 1.5462 respectively.

3.3.4 Path Diversity

The concept of *path diversity* is related to the topic of state-lattice planning described in Section 2.4.1. It refers to the need for the path primitives in the state lattice to be sufficiently diverse, such that plans built using these path segments have the potential to adequately cover the entire state space. For example, if only left turning motions were available, there might be regions of the state space that become unreachable with only these primitives. This would imply a low path diversity.

Knepper & Mason (2009) and Erickson & LaValle (2009) describe various techniques to generate sets of path primitives including; the Green-Kelly approximate area metric, the Hausdorff metric, the mutual-collision metric, and others. Both works conclude that uniform sampling is ineffective at ensuring path diversity, but Knepper & Mason go further by showing that path diversity is not the only issue affecting the successful performance of planners using sets of path primitives.

Both authors note that in general, as the number of primitives used in the set increases, t_P increases and t_E decreases. Knepper & Mason (2009) also note the counterintuitive result that their full path set of 2401 primitives performed worse than various metric-based sets with around 50 primitives, in one particular experiment. Regardless of the effect on t_P and t_E, it is clear that adjustment of both the number of primitives, and the method of their selection from the full control space of the agent, are tuning parameters that can affect the total time required to achieve a goal.

Results demonstrating the effect of different levels of path diversity may be found in Howard et al. (2008), Erickson & LaValle (2009), and Knepper & Mason (2009).

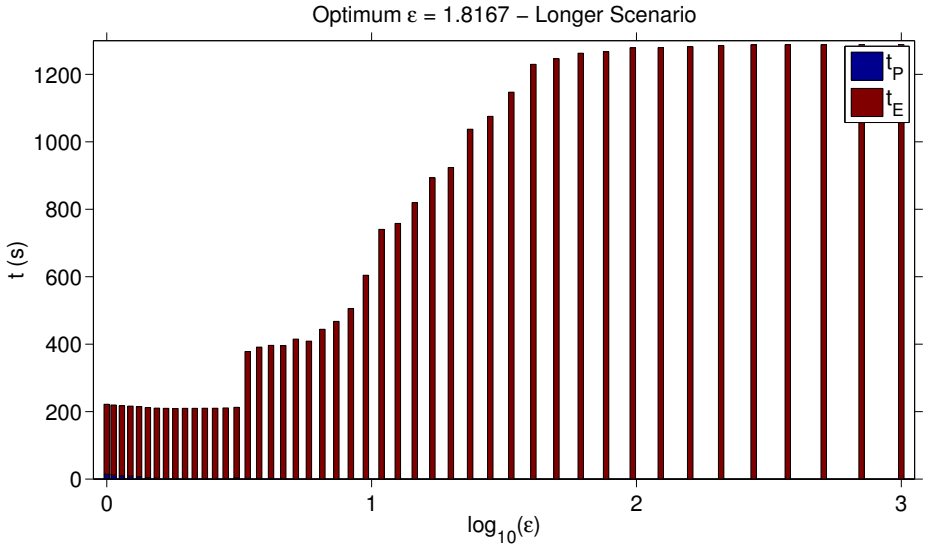


Figure 3.6: Total time required to achieve a goal in the longer static planning scenario versus the heuristic inflation factor, ε, using the A^* algorithm. The minimum total time for this scenario and parameter set is $209.2s$, and is achieved when $\varepsilon = 1.8167$.

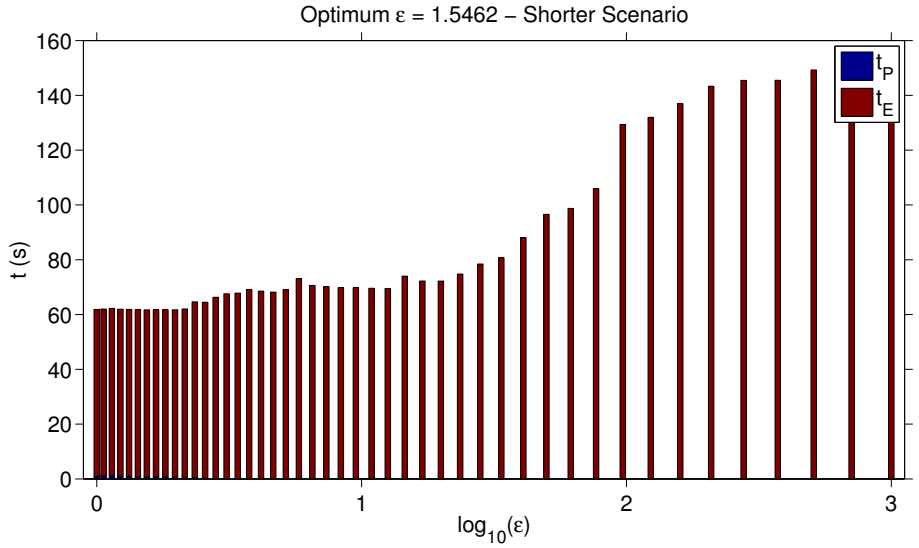


Figure 3.7: Total time required to achieve a goal in the shorter static planning scenario versus the heuristic inflation factor, ε, using the A^* algorithm. The minimum total time for this scenario and parameter set is $61.7s$, and is achieved when $\varepsilon = 1.5462$.

3.3.5 Other Parameters

The previous four sections showed examples of the effect of various tuning parameters that are either generally applicable, or relevant to commonly used algorithms or domains. There are many less general parameters that occur in specific scenarios, that can also affect the trade-off between t_P and t_E.

One example that occurs in hierarchical systems is the scheduling of higher and lower level tasks. In the HR for example, the size of the local planning region affects the planning time required by the local level. This then affects the overall performance because as this region increases in size, the plans are closer to the global optimum, but are refreshed less frequently. As shown by Equation 2.2, there is some flexibility in the constraints on planning time, and thus the size of the local planning region is a suitable tuning parameter for the HR system. Similar parameters can be found in the hierarchical systems of Blythe & Scott-Reilly (1993), where adjusting the time limit for reactive demands before a deliberative plan is computed has an equivalent effect, and Jensen & Veloso (1998), where changing the slope of the required planning time versus environment discretisation curve has the same performance trade-offs.

In some planning frameworks, such as the HR and PHR, multiple algorithms and data structures are available. If these choices can be made just before run-time (or even during run-time by the TOPE process) then they are effectively another tuning parameter. For example, as Figure 2.7 illustrated, the D^* algorithm can have worse performance than A^* when the state space changes near the planning algorithm's goal. If it is known that such situations may occur in the planning scenario, then switching to an alternative algorithm could be beneficial. Similarly, it may be known that an RRT algorithm could perform better than a state-lattice planner in complex environments. In general, building a framework that supports multiple planners and data structures is a significant undertaking, but the potential for performance improvements is also significant.

3.3.6 Summary

This section presented a selection of possible tuning parameters and categories thereof, that were able to affect the trade-off between t_P and t_E. A general design methodology was presented, which required a repeatable simulation environment and a static state space for

the scenario. A brute-force process was used to perform planning and execution for each value in the space of available parameters, and the system metric was applied afterwards to determine the parameter value with optimum performance. This process was used to generate Figures 3.1 through 3.7.

When multiple parameters are present and can be adjusted simultaneously, the space of possible sets of parameter values grows significantly, but the brute-force approach is still possible since it can be done offline. Figure 3.8 shows the surface of total times required to achieve a goal for the combination of grid cell size and heuristic weighting parameters from Figures 3.1 and 3.6 respectively. The figure shows that for the longer static planning scenario and the discrete sets of parameter options, the combined parameter set that results in the minimum total time to achieve a goal has a heuristic weighting of 1.9745 and a grid cell size of $0.75m$. Similarly, Figure 3.9 shows this surface for the shorter static planning scenario, and shows that the best performing parameter set has a heuristic weighting of 2.3462 and a grid cell size of $0.25m$. As before, plans which required longer than $15s$ to compute are not shown in the data, since although they may further reduce the expected execution time compared to other options, they do not further reduce the total time. Also as before, for cell sizes of $6m$ and $8m$ in the longer scenario, the goal location is considered to be an obstacle region and thus the A^* search fails immediately.

The problem with such an approach is that variations in the scenario have an effect on the optimum choice of parameter, and an even more significant effect if there are multiple parameters and the optimum combination is required. In a dynamic state space, there are changes whenever new information is incorporated, hence the optimum parameter(s) can be invalid almost immediately. The TOPE process presented in Chapter 4 and implemented in Chapter 5 estimates the optimum parameter(s) at every planning step in the replanning process, and is able to obtain the benefits of near optimal tuning for every plan.

3.4 Aborted Anytime Planning

This section describes a novel technique to determine when to abort the deflation process within an anytime planning algorithm. The technique relies on the fact that when using an anytime algorithm, the inflation factor, ε, provides an upper bound on the cost of the solution as compared to the optimum (minimum) cost. However, this optimum cost is never

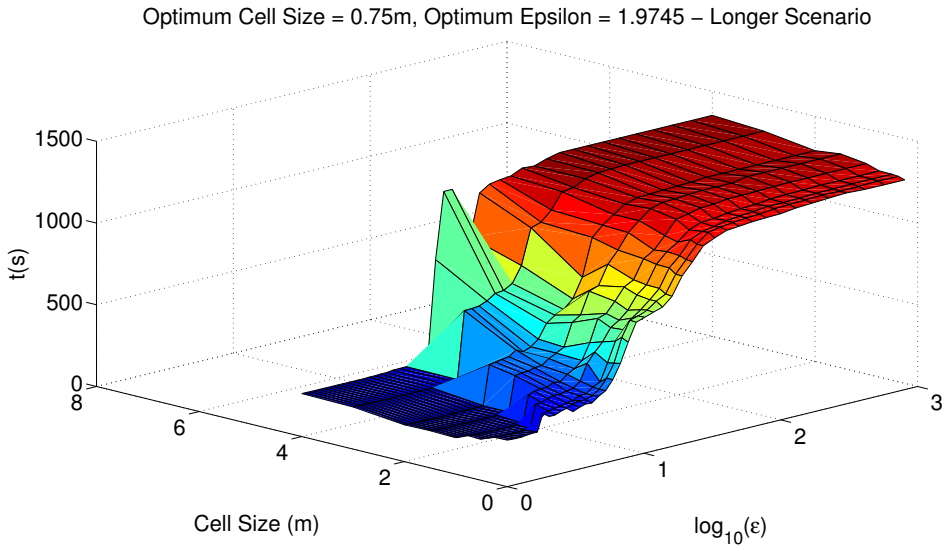


Figure 3.8: Total time required to achieve a goal in the longer static planning scenario versus the choice of heuristic weighting and grid cell size, using the A^* algorithm. The minimum total time for this scenario and parameter space is $201.9s$, and the optimum parameter set has a heuristic inflation factor of 1.9745 and a cell size of $0.75m$.

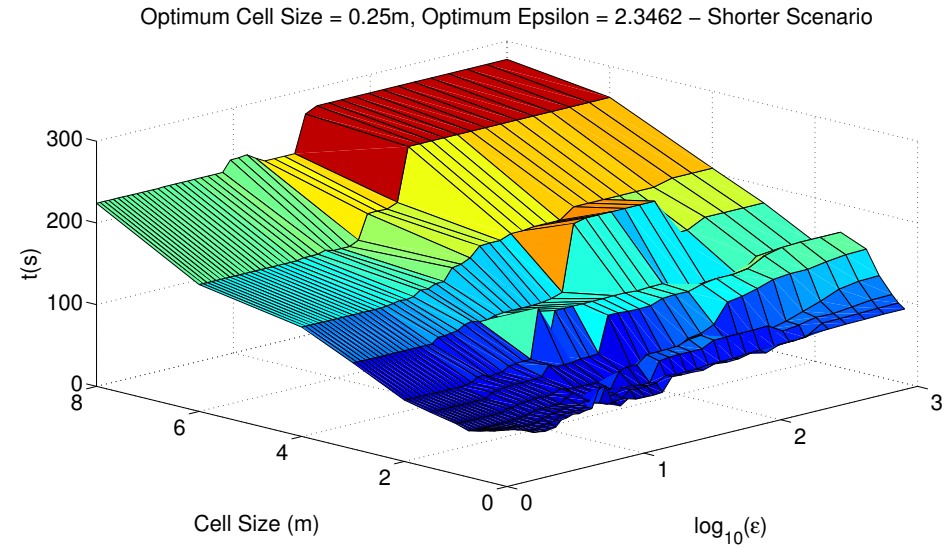


Figure 3.9: Total time required to achieve a goal in the shorter static planning scenario versus the choice of heuristic weighting and grid cell size, using the A^* algorithm. The minimum total time for this scenario and parameter space is $48.3s$, and the optimum parameter set has a heuristic inflation factor of 2.3462 and a cell size of $0.25m$.

known until the planner deflates to $\varepsilon = 1$, a value which may be computationally infeasible to achieve. This section demonstrates that there exists a simple solution to continually bound the optimum cost, and to do so in a manner which provides an admissible heuristic for this optimum cost, since it is shown to never overestimate.

After the i-th iteration of an anytime search algorithm in a static state space, a plan has been computed with cost c_i, given a heuristic inflation factor of ε_i. It is shown in (Pearl 1984) that a weighted graph search plan with weighting factor ε results in a cost, $c \leq c^* \times \varepsilon$, and each iteration of an anytime algorithm can be considered as a single weighted graph search plan. Given this, it can be deduced that the optimum cost in any anytime iteration, i, is:

$$c_i^* \geq \frac{c_i}{\varepsilon_i} \tag{3.3}$$

In subsequent iteration $i+1$, with $\varepsilon_{i+1} < \varepsilon_i$, the cost c_{i+1} could potentially be higher, lower, or equal to c_i (since anytime planning only reduces the upper bound on the solution cost in each iteration, not necessarily the cost). As a result, the newly determined lower bound on optimal cost could be higher or lower than the previous, but is only maintained if it is higher. Thus the lower bound on optimum cost as determined after iteration i is:

$$c_i^* = max \left\{ c_{i-1}^*, \frac{c_i}{\varepsilon_i} \right\} \tag{3.4}$$

with $c_0^* = -\infty$. This approach provides a non-decreasing estimate of the lower bound on solution cost. The following theorem shows that c_i^* will also never overestimate c^*.

Theorem 1. *For all i, $c_i^* \leq c^*$.*

Proof.
After iteration n with $\varepsilon_n = 1$, $c_n = c^*$ by definition. Since:

$$c_n^* = max \left\{ c_{n-1}^*, \frac{c_n}{\varepsilon_n} \right\} \tag{3.5}$$

multiplying both sides by ε_n gives:

$$c_n^* \times \varepsilon_n = max \left\{ c_{n-1}^* \times \varepsilon_n, c_n \right\} \tag{3.6}$$

However, $\varepsilon_n = 1$, and $c_n = c^*$, thus:

$$c_n^* = max\left\{c_{n-1}^*, c^*\right\} \tag{3.7}$$

Since the sequence of c_i^* is non-decreasing for all i and true for $i = n$, the theorem holds[6]. □

Figure 3.10 illustrates this bounding process for a static planning scenario, similar to those used in the previous section. The planning system uses the Anytime-D^* algorithm, with a cost grid cell size of $0.25m$, and ε deflated from 4.0 to 1.0. The planning system also uses a time-focussed metric (with velocity limits determined from the cost map, as before) meaning that the cost is the expected execution time.

The figure shows the expected execution time as determined by each individual plan in blue, versus the cumulative planning time taken by the anytime deflation process. The black line shows the lower bound on this expected execution time as determined by the bounding process, versus the cumulative planning time. The dotted horizontal line shows the true optimum expected execution time, which is only found once the plan with $\varepsilon = 1$ is completed, after a total delation process time of $56.1s$. All of these measures are against the left vertical axis, which is in units of expected execution time in seconds. The cost bound is seen to change to a new value, greater than or equal to the previous value, whenever a plan with reduced ε is computed. The time at which each subsequent plan is fully computed is shown by a black circle on the cost bound line.

If a planning system uses a time-focussed metric (as in the figure), this bounding process can be used to determine when to abandon further anytime deflation. After the i-th iteration of an anytime search algorithm, a plan has been computed in time t_{P_i} and with expected execution time $\hat{t_{E_i}}$, given suboptimality bound ε_i. As given by Equation 3.4, an admissible lower bound for the optimal expected execution time $\hat{t^*_{E_i}}$ is also available. Since the goal is expected to be acquired after a further $\hat{t_{E_i}}$ of time, anytime planning should be abandoned when:

$$t_{P_{i+1}} \geq \hat{t_{E_i}} - \hat{t^*_{E_i}} \tag{3.8}$$

The difference between the expected execution time of the i-th plan, and the highest known lower bound on the optimal expected execution time, represents the possible benefit from

[6]This proof was originally published in Allen (2010) in a different form, with some ambiguity. The version in this thesis has been corrected.

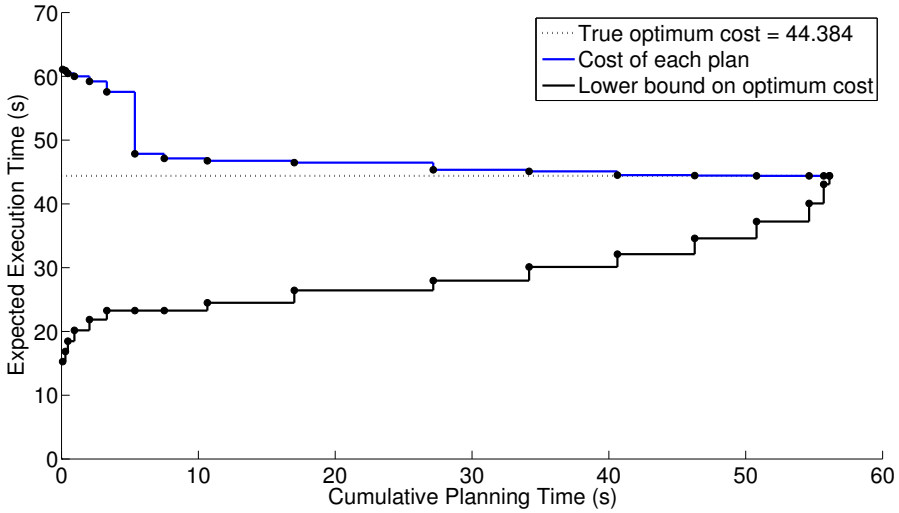


Figure 3.10: Illustration of the process of calculating an admissible bound on the expected optimal cost available from an anytime algorithm. The optimum expected execution time is 44.4s, and is found after the plan with $\varepsilon = 1$ is completed, after a total delation process time of 56.1s.

continued planning. If computation requires more time than will be saved at execution, it is not worth continuing to plan. A planning system which incorporates this check can save up to $\hat{t^*_{E_i}}$ of planning time, which can potentially be reinvested into other tasks. Compared to a traditional anytime algorithm, this strategy yields the same total time to achieve a goal, but requires less planning time to do so if $\varepsilon_i > 1$ for all i.

This strategy is suited to static state spaces where planning is first performed to completion, and then execution is initiated[7]. With one minor amendment, it is also suitable to replanning scenarios where planning and execution are performed simultaneously. At each iteration, i, the amount of time for which the previous plan was actually executed, $t_{e_{i-1}}$, is subtracted from the bound on expected optimal execution time. Equation 3.4 thus becomes:

$$\hat{t^*_{E_i}} \geq max \left\{ \hat{t^*_{E_{i-1}}} - t_{e_{i-1}}, \frac{\hat{t_{E_i}}}{\varepsilon_i} \right\} \tag{3.9}$$

with all c replaced by $\hat{t_E}$. Since each plan is executed for only the amount of time until the

[7]This is the type of system used to produce the data visualised in Figure 3.10. In practice, this would never be appropriate, since it would be faster to compute only the plan with $\varepsilon = 1$, avoiding the overhead associated with the rest of the deflation process.

next plan is available, that is:

$$t_{e_{i-1}} = t_{P_i}, \tag{3.10}$$

this gives:

$$\hat{t^*_{E_i}} \geq max \left\{ \hat{t^*_{E_{i-1}}} - t_{P_i}, \frac{\hat{t_{E_i}}}{\varepsilon_i} \right\}. \tag{3.11}$$

This is valid even for agents that suffer imperfect execution – meaning they may be incapable of executing a plan exactly as predicted[8] – because the cost function is employed in the anytime algorithm by means of an admissible heuristic, which is then inflated. The optimum expected execution time, $\hat{t^*_E}$, is an underestimate, as required, and any imperfect execution can only result in an increase in execution time. Thus $\hat{t^*_{E_{i-1}}} - t_{P_i}$ is still guaranteed to be an admissible lower bound, and cannot overestimate.

Unfortunately, this strategy is not applicable to dynamic state spaces, for two reasons. Firstly, as information changes in the state space the optimum cost estimate must be discarded. This is due to the fact that the changes could result in the optimum path having higher or lower cost, and thus the sequence of c^*_i cannot be guaranteed to be non-increasing. Although the sequence could be made non-increasing by setting $c^*_i = -\infty$ every time information in the state space changes, it is likely that the rate at which information changes is higher than the planning rate, making the strategy redundant. Secondly, in a dynamic state space imperfect execution of a plan could have positive effects, and potentially reduce the time required to achieve a goal. Again, this means the sequence of c^*_i cannot be not guaranteed to be non-increasing, making the strategy unsuitable for dynamic state spaces.

Despite this, the technique to calculate an admissible lower bound on the solution cost of an anytime plan is novel, and has use in applications such as operator displays that indicate the expected time remaining to achieve a goal, or estimation processes in which knowledge of the maximum benefit to be obtained may be relevant.

3.5 Conclusion

This chapter has described a wide range of techniques that have the potential to reduce the total time required to achieve a goal, by way of reducing t_P, t_E, or both. Many of these

[8] Formally, this is the case when $\hat{c_E} < c_E$, where $\hat{c_E}$ is the expected cost of execution as determined by the internal metric of the planning algorithm, and c_E is the actual cost of execution determined once a goal is achieved.

techniques have seen extensive use in various fielded planning systems, yet all feature the same limitation – they are tuned for a particular situation, and can perform significantly worse when the situation changes.

Time-optimal planning is computationally infeasible in a dynamic environment which requires replanning. Parallelisation of hierarchical planning systems can be particularly effective if all levels meet the criteria for sufficient reactivity, but can perform up to as poorly as an un-parallelised system if the situation prevents these criteria being met. Tuning parameters set at design time are appropriate for the particular scenario under which they were determined, but may be suboptimal in alternative situations. The aborted anytime planning technique was also shown to be unsuitable for dynamic state spaces.

The following chapter presents the TOPE problem in more detail, and describes the TOPE process which solves this problem at each iteration of a replanning system. Unlike all the techniques presented in this chapter, the TOPE process responds to the changing situation by adjusting system parameters to obtain the best results. It builds primarily on the topics of parallelisation and tuning parameters described in Sections 3.2 and 3.3, as the process estimates the optimal parameter tuning at every iteration of the replanning process, whilst executing the previous plan in parallel.

Chapter 4

The Time-Optimal Planning and Execution Process

The purpose of this chapter is to derive the equations that specify the TOPE problem, and demonstrate their use in the TOPE process; a parallelised replanning process that attempts to minimise the total time required to achieve a goal. Section 4.1 presents this derivation, showing that the objective of a single step of the TOPE process is to determine the optimal set of parameters that are then applied to the replanning system. Section 4.2 describes the TOPE parameter space from which this optimal set is obtained, and presents measures of parameter utility which can be used to determine the appropriateness of particular parameters.

Section 4.3 performs experiments to assess the performance of the TOPE process in comparison to existing methods that use only fixed parameter values throughout the replanning process. The hypothesis is that if the TOPE process can be performed with perfect accuracy and sufficiently quickly, then the resulting total time to achieve a goal will be lower than any of these alternative methods operating in the same parameter space. In order to achieve perfect accuracy, a general procedure is described by which the process can be simulated with high fidelity. The optimum parameter values are determined by brute-force through the entire parameter space (much like the process used in Section 3.3, but at every iteration of the replanning process). Regardless of the time required to perform this brute-force parameter search, it is simulated as taking only a small fixed time, which is adjusted as the independent variable in the experiments. The point at which the TOPE

process results take a greater total time to achieve a goal than the comparison methods then defines how 'sufficiently quick' the process must be.

Where the experiments in the Section 4.3 were designed to determine the required timeliness of the TOPE process for it to outperform alternative methods, the experiments in Section 4.4 show that timeliness can be traded for accuracy. A general procedure is described to analyse the sensitivity of the TOPE process to these factors, by way of a perturbation analysis applied to the parameter value estimates and the estimation time. Experiments are performed that apply this procedure, with the hypothesis being that there exists a suitable operating region wherein sufficient timeliness and accuracy are achievable. Within this operating region, the TOPE process can reduce the total time required to achieve a goal, compared to alternative methods.

Finally, Section 4.5 concludes with a discussion of the main contributions of the chapter, and their application to the more general TOADM problem. The equations that govern the TOPE process are easily extended to the TOADM process, and the general procedures to assess the required accuracy and timeliness of the process are shown to also apply.

4.1 Derivation of the TOPE Equation

Figure 4.1 presents a simplified view of a typical replanning process, similar to Figure 2.9 but making the assumption that a suitable commitment technique is in place, as discussed in Section 2.5.4. Although the figure is presented as a 2D path planning problem for clarity, the analysis and motivation are valid for other planning applications. In the figure, solid red lines are the agent's current plan, dashed red lines are previous plans, and the dotted black line is the path that has been executed by the agent, which is represented by the black triangle.

The first panel shows the initial state of the scenario; no plan has been computed, and the agent is at its initial position. The second panel shows the state after the first plan, P_1 has been computed, taking time t_{P_1}. The agent then starts executing this first plan while the second plan is being computed. The third panel shows the state after this second plan, P_2 is complete, after planning for t_{P_2} and executing P_1 for this same duration. The fourth panel shows the final path of execution, and the total elapsed time.

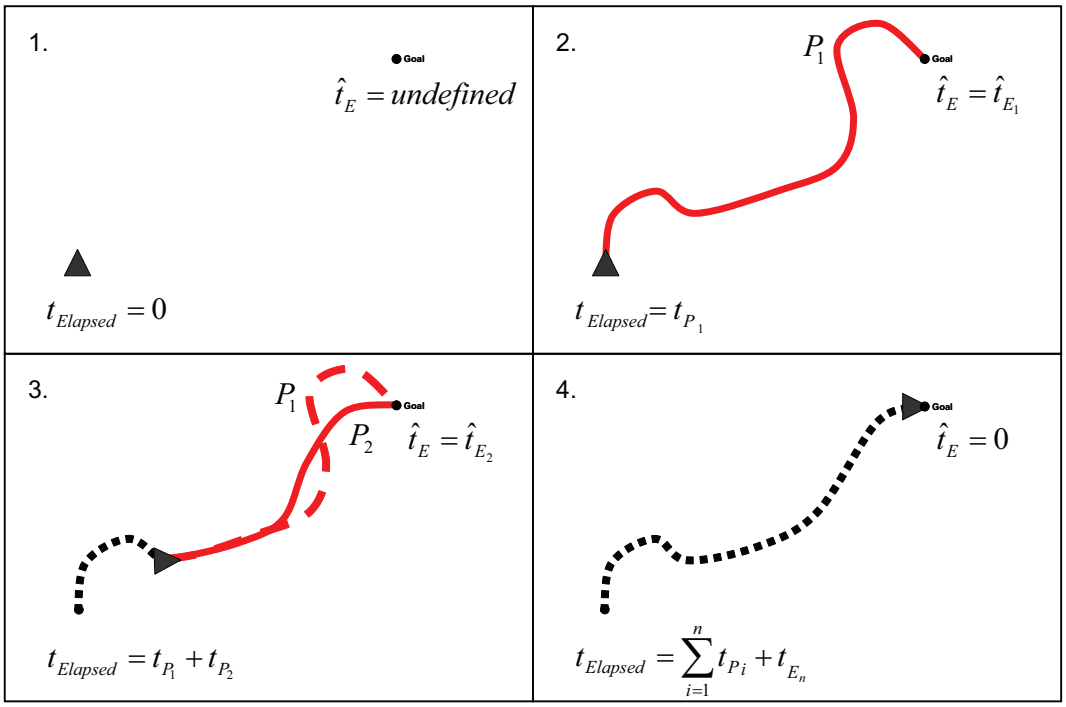


Figure 4.1: Simplified time-series depiction of a typical replanning process in a dynamic state space.

The figure shows that the total time to achieve a goal for a replanning system is given by:

$$t = \sum_{i=1}^{n} t_{P_i} + t_{E_n} \tag{4.1}$$

where t_{P_i} is the time taken to compute the plan to a goal in any iteration, i, and t_{E_n} is the time taken to execute the final plan (where $t_{E_n} < t_{P_{n+1}}$ or else n is not the final iteration). Given that the system cannot know which iteration is the final, in any particular iteration it can only calculate the expected remaining time to achieve a goal:

$$\hat{t} = \hat{t_P} + \hat{t_E} \tag{4.2}$$

(since this is true in any iteration, the i-th iteration labels are removed.)

A single step of the TOPE process invests some initial time, t_C, in order to determine the choice of parameters to the planning system that minimise this $\hat{t}$. The expected total time

in any iteration thus becomes:

$$\hat{t} = t_C + \hat{t_P}' + \hat{t_E}' \tag{4.3}$$

where $\hat{t_P}'$ and $\hat{t_E}'$ are the new expected planning and execution times, given these parameters. If:

$$t_C < \left(\hat{t_P} - \hat{t_P}'\right) + \left(\hat{t_E} - \hat{t_E}'\right) \tag{4.4}$$

then this investment has reduced the total time required to achieve a goal, compared to using the unmodified set of parameters in this iteration, i. The TOPE process performs this calculation of appropriate system parameters in every iteration of the replanning process.

Figure 4.2 redraws Figure 4.1 to instead show the first two iterations in a typical TOPE process for a replanning scenario. Each iteration involves the determination of the appropriate parameters for a plan, the computation of this plan, then the execution of this plan whilst the first two steps are being performed for the next iteration. The durations of time required by the estimation and planning steps in an iteration, i, are denoted by t_{C_i} and t_{P_i}, respectively. The expected time to execute the i-th plan, P_i, is denoted $\hat{t_{E_i}}$. The text in the lower left corner shows the accrued time at each point in the process, and the text under the goal indicator shows the expected execution time remaining.

The diagrams show the elapsed time at each iteration, from which it is determined that the total time to acquire the goal is given by:

$$t = t_{C_1} + t_{P_1} + \sum_{i=2}^{n} [t_{C_i} + t_{P_i}] + t_{E_n} \tag{4.5}$$

where the first two terms represent the time spent waiting until the first plan is available, the middle term represents the total time spent executing each intermediate plan (for $i < n$), and the final term represents the time spent executing the final plan (where $t_{E_n} < t_{C_{n+1}} + t_{P_{n+1}}$ or else n is not the final iteration).

At each iteration in the replanning process, the TOPE problem is defined as:

$$y^* = \arg\min_{y \in Y} \{t\} \tag{4.6}$$

where y^* is the set of parameters to a planning system (in the space, Y, of available parameter sets) which minimises the total time required to achieve a goal, t, given by Equation

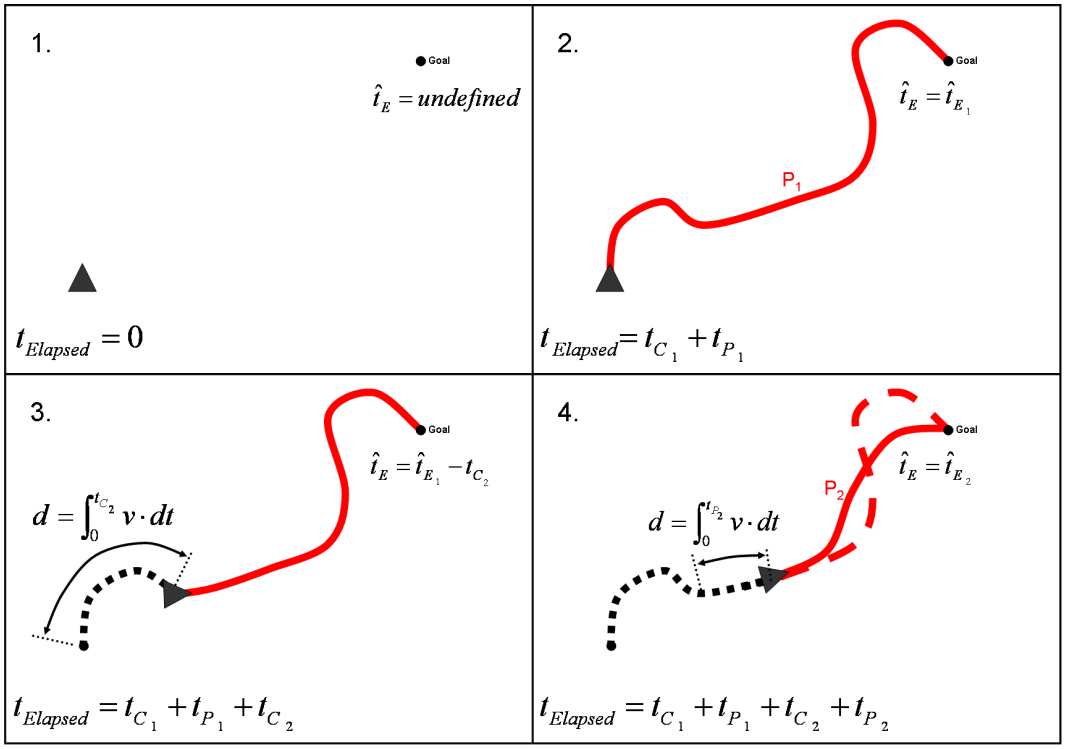


Figure 4.2: Time-series depiction of a typical TOPE process in a dynamic state space involving simultaneous replanning and execution.

4.5. The space Y is application specific but can include any parameter which affects the total t. For replanning systems, these include those parameters described in Section 3.3, and TOPE-specific parameter spaces that are discussed further in Section 4.2.

There are two important points to note regarding these equations. Firstly, Equation 4.6 is used at every iteration, and thus in any iteration j where $i < j$, t_{C_i} and t_{P_i} are constants (since these time periods have elapsed). They are thus independent of Y and can be removed from the equation. Secondly, the true value of Equation 4.5 cannot be known until the goal is acquired, but it can be estimated instead. At any iteration j, this estimate is the sum of the known elapsed time and the expected remaining time. This latter point implies that:

$$\hat{t_{E_j}} \equiv \sum_{i=j+1}^{n} \{\hat{t_{C_i}} + \hat{t_{P_i}}\} + \hat{t_{E_n}} \tag{4.7}$$

That is, since the expected future time periods (or even the number of them, n) cannot be known until they have elapsed, it is valid to replace their sum by the best estimate of the

expected remaining time in any iteration; $\hat{t_{E_j}}$.

Putting both these points together, and substituting Equation 4.5 into Equation 4.6 gives the TOPE equation at the beginning of any iteration, j, as:

$$\begin{aligned}
y^* &= \arg\min_{y \in Y} \left\{ t_{C_1} + t_{P_1} + \sum_{i=2}^{n} [t_{C_i} + t_{P_i}] + t_{E_n} \right\} \\
&= \arg\min_{y \in Y} \left\{ \sum_{i=j}^{n} [t_{C_i} + t_{P_i}] + t_{E_n} \right\} \quad \text{(Removing constant terms with } i < j\text{)} \\
&= \arg\min_{y \in Y} \left\{ t_{C_j} + t_{P_j} + \sum_{i=j+1}^{n} [t_{C_i} + t_{P_i}] + t_{E_n} \right\} \quad \text{(Extracting } j\text{-th terms)} \\
&\equiv \arg\min_{y \in Y} \left\{ \hat{t_{C_j}} + \hat{t_{P_j}} + \sum_{i=j+1}^{n} \left[\hat{t_{C_i}} + \hat{t_{P_i}}\right] + \hat{t_{E_n}} \right\} \quad \text{(Taking expected values } \forall i \geq j\text{)} \\
&= \arg\min_{y \in Y} \left\{ \hat{t_{C_j}} + \hat{t_{P_j}} + \hat{t_{E_j}} \right\} \quad \text{(Substituting Equation 4.7)}
\end{aligned} \tag{4.8}$$

This derivation yields the final form of the TOPE equation:

$$y^* = \arg\min_{y \in Y} \left\{ \hat{t_C} + \hat{t_P} + \hat{t_E} \right\} \tag{4.9}$$

where the time components are the expected times required to compute the solution to Equation 4.9, compute the plan given the argument set y^*, and execute the computed plan, respectively. This is the general form of the TOPE equation, and the j-th iteration labels are removed because it is applicable at all iterations.

Succinctly, the TOPE equation states that the best parameters to apply to the replanning system are those that are expected to minimise the total time required to achieve a goal. Importantly, these are not the parameters that minimise $\hat{t_P} + \hat{t_E}$; they are the parameters that minimise $\hat{t_C} + \hat{t_P} + \hat{t_E}$, meaning the parameters account for the time required to find them. The TOPE process using this equation can be considered to be a statement of the *dynamic programming principle* (Bellman 1957*a*); when the future effects of decisions are uncertain, by taking the optimal action at each point in time, the resulting behaviour will be as close to globally optimal as possible. The TOPE equation is analogous to a *policy* used by an MDP (Bellman 1957*b*), or the Bellman equation (Bellman 1957*a*, chap.III). The

major difference is that where MDPs or dynamic programming using the Bellman equation require a probability function on the outcome of applying an action to a state, the TOPE process assumes that this probability function is unattainable, because the state transitions are affected by an unknown and possibly non-stationary stochastic process.

Perhaps the simplest summary of the TOPE process can be found with reference to Theodore Roosevelt's quotation from the preface of this thesis:

> "In any moment of decision the best thing you can do is the right thing. The worst thing you can do is nothing."

The TOPE process tries to do the right thing at every moment of decision, but with the knowledge that taking too long to decide it is the worst possible thing.

4.2 The TOPE Parameter Space

The output of the TOPE equation is a set of parameters, the y^* of Equation 4.9, from a problem-specific parameter space. This space can include any continuous or discrete parameter which effects a change in the total time required to achieve a goal, by altering the trade-off between t_C, t_P, and t_E. Section 3.3 described several such parameters including; state space discretisation parameters, sampling density, heuristic weightings, path diversity, hierarchical scheduling parameters, and the choice of planning algorithms. This list was not exhaustive, but gave an indication of the scope of planning problems for which the TOPE process could be applicable.

This list only featured parameters that affected t_P and t_E, but parameters that affect t_C can also affect the total t. This is because t_C, the time taken to compute the resulting y^* of Equation 4.9, is itself a variable within the equation (more accurately, its expected value, $\hat{t_C}$, is the variable in question). Parameters that affect the expected value of t_C can thus affect the total time required to achieve a goal. Discussion of the types of TOPE estimation techniques is left until Section 5.1, but a general procedure that searches for particular parameters within some space, can be expected to require computational time that is affected by the size and distribution of this space. Thus, t_C is likely to be proportional to the size of Y, or even the number of sets of combinations of parameters $y \in Y$.

Given this dependence, the level of discretisation of continuous parameters that are assessed by the TOPE estimation process will have an effect on t_C. For example, reducing the number of possible heuristic weightings or grid resolutions reduces the size of Y, and could be expected to reduce t_C. However, the distribution of Y will also alter the effect of adjusting the level of discretisation. In this example, if the most beneficial heuristic weightings or grid resolutions are removed from the available choices, then t_P and t_E may also be affected. With knowledge of the distribution of Y, and also the relative utility of each of the parameters therein (discussed in Section 4.2.1 below), altering the discretisation of each parameter can be used as a tuning parameter for the TOPE process.

Similarly, since the distribution of Y is important, the choice of parameters to ignore is also a tuning parameter. For instance, ignoring a subspace of Y because it would increase t_C excessively and render the TOPE process computationally infeasible. Alternatively, if a planning system already had anytime capabilities, it might be reasonable to ignore the set of possible heuristic weightings in the TOPE estimation process, and to spend the available estimation time on other subsets of Y.

4.2.1 Tuning Parameter Utility

Parameter utility is a measure of the effectiveness of a tuning parameter as a means of affecting the total time required to achieve a goal, for a particular planning system. Parameters take either continuous or discrete values, over ranges from few to many orders of magnitude, and have drastically different effects on system performance. Accordingly, parameter utility is an imprecise measure, but can still be a valuable design tool. This is particularly true when there are many possible tuning parameters, and the choice of which to ignore or restrict is necessary in order to limit estimation time to a practical level.

There are many factors that affect parameter utility, with one of the most significant being the effect of this parameter on the total time required to achieve a goal, as it is varied over its full range. This is the easiest factor to assess however, since it can be performed by an exhaustive simulation of the planning system using every possible parameter value for a static scenario. This was the technique used to generate all the figures shown in Section 3.3, and is used to determine the optimum *a priori* parameter values by many existing systems. It is important to note that this is not necessarily a good indication of

the maximum improvement available from the use of a parameter in the TOPE process, since the dynamics and structure of the state space can significantly alter the shape of these graphs at each iteration of a replanning process.

A second factor affecting parameter utility is the rate and magnitude of variation in the optimum value of the parameter between iterations of the TOPE process. This can be measured by reporting a 'trace' of the estimated optimum parameter values as the TOPE process is run (as described in Section 4.3.5), and applying a Fourier transform to determine the frequency response. For comparison between multiple parameter sets, the TOPE process should be applied over the full parameter space and simultaneous traces can compare the rate and magnitude of variation of all parameters. If a parameter is found to vary insignificantly, this suggests it has a low utility, and could perhaps be left constant and removed from Y. As noted above however, this is again not necessarily a good indication of the potential utility in every scenario, since the dynamics and structure of the state space affect the resulting traces.

A third factor affecting parameter utility is the sensitivity of TOPE performance to inaccuracy in its estimation, which must be measured in a replanning scenario. The full procedure for this technique is described in Section 4.4.1, and provides two measurements useful for assessing parameter utility. The first is the slope of the mean TOPE performance curve as the amount of variation in the parameter increases, which indicates how accurately the parameter must be estimated before performance is negatively affected. The second is the variance of this curve, which indicates the volatility of the sensitivity to the parameter. Parameters with high volatility might need to be estimated more accurately to meet desired confidence levels for TOPE performance.

All these factors must be tempered with a measure of the suitability of the parameter to the system's implementation. A parameter can have low practical utility despite appearing to be valuable by the three previous measures, if it is unsuitable for implementation in an existing system. As described earlier, part of the TOPE process involves moving parameters that were traditionally fixed at design time, and allowing them to be adjusted at runtime. If a parameter is unsuitable for run time adjustment, then it has a low practical utility for the system. An example of this might be state space discretisation parameters, since one method of allowing adjustment at runtime involves maintaining multiple data structures with different discretisations, thus duplicating data and requiring extensive management

of its synchronisation. Such changes might prove impossible to undertake in an existing system, giving the parameter a low utility.

Parameter utility as a design tool is of most use when there are many parameters that could be employed in the TOPE process, and selection of those with the highest utility is required. For an existing system with no runtime parameters, it is likely that bringing the easiest parameters to implement to a state of runtime adjustability will be of greater benefit than attempting to determine the most useful from static analysis alone.

4.3 Application of the TOPE Process

This section describes the use of the TOPE process in a replanning system, and performs experiments using the CORD vehicle simulation in the shorter 2D planning scenario shown in Figure 2.2 (and reprinted as Figure 4.3 below). The use of the TOPE process is presented as a general procedure in Section 4.3.1, with a brief discussion of the effect of running it in series or in parallel with the planning process. Section 4.3.2 describes the methodology behind the experiments performed in this section. These experiments aim to show that the TOPE process has the potential to result in a lower total time to achieve a goal than alternative methods described in Section 4.3.3. The experiments are performed using two example TOPE parameter spaces; a one-dimensional space in Section 4.3.4, and a two-dimensional space in 4.3.5.

4.3.1 General Procedure

Figure 4.2 illustrated a time-series depiction of the TOPE process. The procedure is similar to that performed by the highest level of the hierarchy in the PHR system, save that the estimation step is performed immediately prior to every plan. This procedure is formalised in Algorithm 4.1, which runs in parallel with any lower levels in the hierarchy and the agent's controllers.

In this procedure, the TOPE process augments the highest planning level in the hierarchy (the 'global' level), computing the resulting y^* of Equation 4.9 and applying these parameters to the system just prior to planning. The effect is that each iteration of this level in the process takes $t_C + t_P$ to complete, rather than just t_P. It should be noted that the

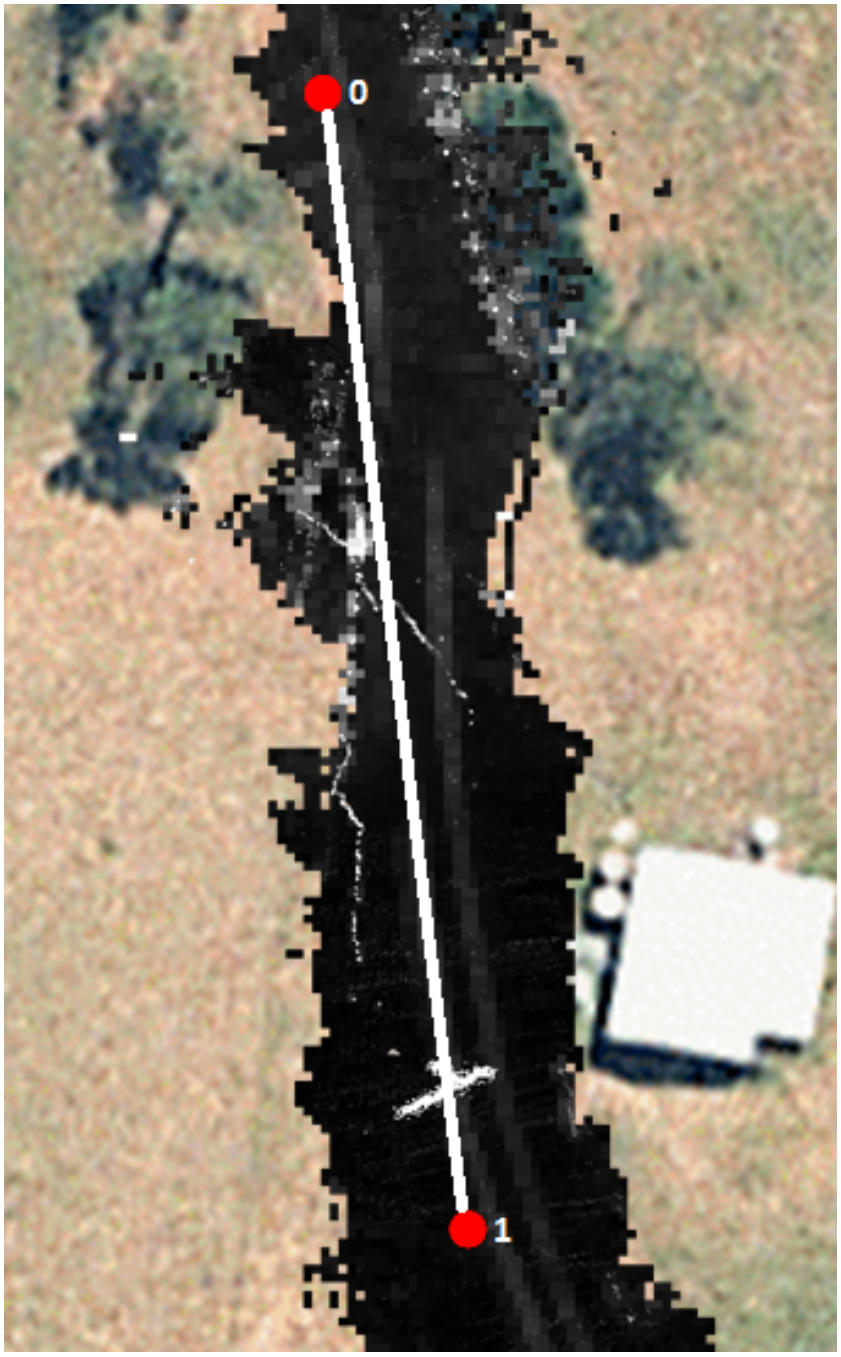


Figure 4.3: The shorter 2D path planning scenario employed in many of the experiments in this thesis (reprinted from Figure 2.2).

Algorithm 4.1 A general procedure to apply the TOPE process within a replanning system.

```
1: while goal has not been achieved by the planning system do
2:    incorporate new state space information
3:    determine y* using Equation 4.9, taking time t_C
4:    apply y* to planning system
5:    compute a new plan from the end of the committed plan to the goal, taking time t_P
6:    if there are lower levels in the hierarchy then
7:       send this plan to the next lower level in the hierarchy
8:    else
9:       determine subset of this plan within commitment length
10:      send this subset to the agent's controllers as the committed plan
11:   end if
12: end while
```

technique used to solve Equation 4.9 in line 3 is unspecified, as there are many possible techniques. Several example implementations are described and evaluated in Chapter 5. The time taken to apply the parameters y^* to the planning system is negligible in many cases, such as if it requires just the manipulation of a value in software. Where this does require some time, such as copying a large quantity of data from a database, the duration can be included within t_C and the TOPE equations and process remain unchanged.

A possible alternative structure for the TOPE process is to separate the calculation of parameters from the planning functions, and to parallelise the two parts. This is justified by the exact same motivation as was used to progress from the HR to the PHR. In that case, it was shown that by allowing the local and global planning levels to operate in parallel, improvements were obtained because the local planner could continue optimising a subset of the full problem even when the global planner was unresponsive. Similarly, when applied to the TOPE process this would allow the global planner to continue optimising its subset of the full problem (albeit with potentially suboptimal parameters), even when the technique used to calculate Equation 4.9 is unresponsive. This possibility is not considered in this thesis, but represents a likely area of future work.

4.3.2 Experimental Method

As described above, the purpose of the experiments in this section is to show that the TOPE process has the potential to result in a lower total time to achieve a goal than alternative methods. The hypothesis is that if the TOPE process can be performed sufficiently fast

and perfectly accurately, then it can reduce the total time to achieve a goal beyond that of any process using only fixed parameter values from the same parameter space. A second hypothesis is that if the dimensionality of the parameter space is increased, the TOPE process can be expected to further improve performance beyond what was possible with only a subset of the parameter space.

The experimental method uses the procedure from Algorithm 4.1, but runs in simulated rather than real time. This is necessary to attain perfect accuracy for the solutions to Equation 4.9, but the method performs a brute-force evaluation of the planning and expected execution times for every parameter set in the parameter space. The time required to perform this *brute-force parameter search* is very high, but the simulation pauses the replanning system while it is performed, and reports it as taking only a low value of t_C. This value is the dependent parameter in the experiments described in Sections 4.3.4 and 4.3.5, and is linearly increased from 0 to $3s$ in increments of $0.2s$.

The experiments were performed on the 2D path planning scenario described in Section 2.2.1 (and reprinted as Figure 4.3 above). Unlike the demonstrations of the effect of varying parameter values shown in Section 3.3 which used the cost grid from this scenario as a static state space, these experiments treat it as a dynamic state space. In the static case, all data in the grid structure was known to the planning algorithms from the start of the process. In the dynamic case, data is loaded from the grid as the agent traverses it, with new data received by the planning algorithms at $100Hz$, and retained thereafter. The data to be loaded is determined by considering there to be a SICK 2D laser scanner mounted at a particular position and orientation on the agent. Ray-tracing geometry is then used to calculate the points on the ground plane that would be illuminated by this sensor, and the grid cells corresponding to these points are loaded and passed to the planning algorithms. Any unknown regions are assumed to be traversable at maximum velocity (whereas there are no such regions in the static case)[1].

The cost function employed by the planning algorithms is the expected execution time,

[1] Although this treatment affects the expected execution time of the planner's output, in practice these areas are always revealed by the simulated agent's sensors before they are ever traversed. As a result, the measured execution time is always correct, despite the planners treating unknown regions optimistically. It is also relevant to point out that the combination of this cost value for unknown regions, and the heuristic function used, tends to reduce the computational time required by the planning algorithms. This is because the heuristic cost is closer to the true cost, which limits the number of states explored by a graph search algorithm (Pearl 1984, Russell & Norvig 2003).

and uses the same velocity profile function described in Section 2.2.1. A suitable heuristic cost function is to assume the agent can move at its maximum velocity along the straight line between two locations, which is an admissible function since the true velocity cannot be greater. The straight line distance between the start and goal points is $128m$, giving a minimum heuristic estimate of traversal time of $25.7s$, given a maximum velocity of $5m/s$. The optimum path (for a grid with cell size of $0.375m$, shown to be the best choice for static scenarios in Figure 3.2) has an expected traversal time of $48.6s$, reflecting the necessary traversal of low velocity regions.

This method of loading data from the stored cost grids and supplying it to the planning algorithms is based on the techniques used in the HR and PHR systems described earlier. The software interfaces used in these experiments to ensure that planning and updating of data are asynchronous, are exactly the same as those used in the real systems. The use of ray-tracing geometry to load data from the regions of the world that would be perceived by a real agent with the same pose further ensures that the operation of the simulated system exhibits equivalent behaviours to the real system. Further details of the planning system used for these experiments are available in Allen et al. (2009) (reprinted in Appendix B).

As with the experiments described in Section 3.3, the experiments were performed on an Intel X86 architecture using standard consumer operating systems. As a result, despite the use of a deterministic simulator which repeatably returns the same plan and execution time for a given scenario and parameter set, the measured planning time is only accurate to within approximately $\pm 8ms$, and is non-deterministic due to the operating system scheduling other tasks outside of the simulator's control.

In order to determine the values of t_C for which the TOPE process results in a lower total time to achieve a goal than alternative methods, the experimental procedure was performed 10 times for each value of t_C. It was noted that fluctuations in the mean and standard deviation values for the total time to achieve a goal were less than 5% of the total value after only 6 repetitions, which justifies the choice of 10 repetitions as sufficient to determine the distributions. Claims that the performance of the TOPE process and other methods are significantly distinct are only made when the 2σ bounds on each method are separated, implying a 95% confidence interval[2].

[2]Approximately 95% of the results will lie within $\pm 2\sigma$ of the mean, under the assumption that the results are drawn from a normal distribution (Phipps & Quine 1998).

4.3.3 Comparison Techniques

The TOPE process is compared against eight alternative methods, which are all examples of existing techniques used in current state-of-the-art planning systems in this and similar scenarios. The techniques are distinguished by number, and given the prefix 'B' as they are the baselines used for comparison in these and later experiments. Four of the techniques also have the suffix 'b' to indicate that they are equivalent to the original techniques, save that they make use of information from a brute-force parameter search in a static state space (specifically, those analysed in Section 3.3). This static space contains all the information that would potentially be available to an agent incrementally perceiving the space. In other words, this reflects the difference between having and not having some previously acquired information about the appropriate tuning parameters to use in this particular state space. It is valuable to examine both cases, since in some decision making scenarios this information is easily acquired or readily available, and in others it can be impossible to acquire.

In all the following descriptions, c refers to the side length of a square grid cell, and ε is the inflation factor applied to the heuristic used by the global planning algorithm. Brief descriptions of each technique are as follows:

- B1 uses the A^* algorithm with a fixed c and $\varepsilon = 1$.
- B2 uses the Weighted A^* algorithm with a fixed c and a fixed $\varepsilon > 1$.
- B3 uses the Anytime-A^* algorithm with a fixed c.
- B4 uses the Anytime-D^* algorithm with a fixed c.

For the shorter scenario, the differences in c and ε values between the two categories of baseline techniques are given in the table shown in Figure 4.4. When prior information is available, the parameters take the the best performing values as found in Figures 3.2, and 3.9, from Chapter 3. Without additional information, the cell size of $1m$ is the value determined empirically in the HR and PHR systems in previous field operations. Similarly, the value of $\varepsilon = 1.5$ used in technique $B2$ without additional information, reflects a decision made in those field operations in which the operators were satisfied with the agent's paths being up to 50% longer, provided the stationary 'thinking time' was reduced.

		Additional Information		No Additional Information	
	Parameters:	c	ε	c	ε
Estimator	B1	0.375	1.0	1.0	1.0
	B2	0.25	2.3462	1.0	1.5
	B3	0.375	variable	1.0	variable
	B4	0. 375	variable	1.0	variable

Figure 4.4: Cell size and heuristic weighting parameter values for the two categories of baseline estimation techniques in the shorter scenario.

The resulting total times to achieve a goal for each of these baseline techniques are shown in Figure 4.5. The results indicate that, as expected, techniques that make use of the best parameters determined from prior information perform better than those without such information.

4.3.4 One Dimensional Parameter Space Example

This section demonstrates the TOPE process for a planning system with a one dimensional parameter space operating in a dynamic state space. The purpose of this experiment is to verify that the TOPE process can achieve a reduction in the total time required to achieve a goal, when compared against other replanning techniques. The experiment uses the state space discretisation parameter space, adjusting the grid cell size into which the underlying cost data is reinterpreted. The specific set of values used is the same as that discussed in Section 3.3, and used in Figures 3.1 and 3.2.

Figure 4.6 shows the results for the state space discretisation parameter space. The results indicate that for this particular scenario and parameter space, the TOPE estimation process has better performance than all the techniques used in the HR and PHR systems[3], for all simulated estimation times up to $3s$. Since technique B1b is the uninflated A^* algorithm using the best known cell size from an offline analysis of the state space, it represents the

[3]The poor performance of the Anytime D^* algorithm in these results is attributed to the large percentage of the nodes in the state space graph that are potentially changed in each sensing step. The nodes in this set must all be checked for variation in each iteration of the algorithm, leading to considerable overhead. The Anytime D^* algorithm is very efficient at deflating the heuristic when this set is empty, and is also efficient at incremental repairs of a path without deflating the heuristic, but can have the poor performance seen in these results when both are attempted simultaneously.

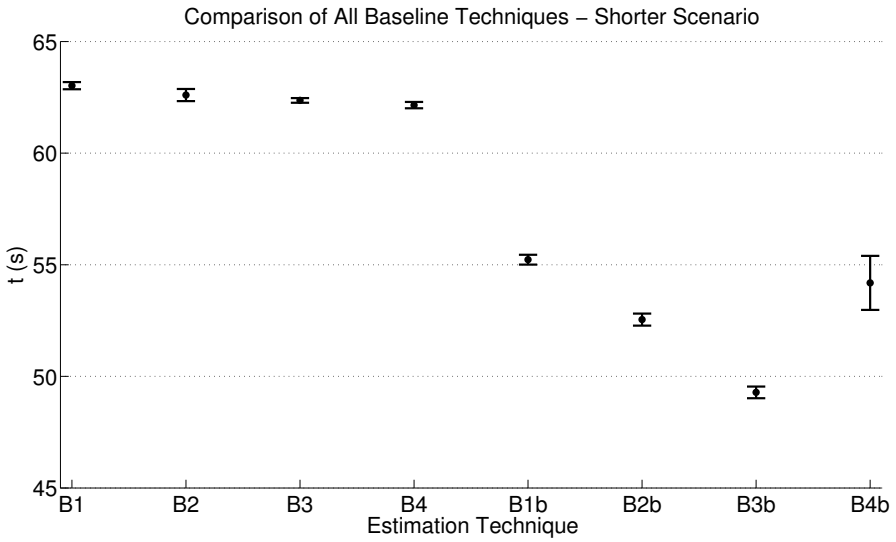


Figure 4.5: Total time to achieve a goal for the eight baseline comparison techniques. The error bars show two standard deviations from the mean after 25 repetitions of each technique.

informed choice of parameters but in the same parameter space (cell sizes only) as the TOPE process and is thus a fair comparison. If the estimation time is within $1.2s$ or less, then the 95% confidence intervals between B1b and the TOPE process are separated. By this criteria it can be said that the TOPE process is capable of superior performance than any fixed parameter choice technique operating in the same parameter space. The three baseline techniques that have superior performance to the TOPE estimation process all adjust the ε value as well, meaning that they have access to additional tuning parameters and are thus not a fair comparison in this experiment.

It should be noted that these numerical values are not indicative of the expected performance in other scenarios or for other parameter spaces. The value of these results is that they verify that the TOPE process has the potential to outperform alternative techniques using fixed parameters provided that the estimation step can be performed sufficiently quickly, and with perfect accuracy.

4.3.5 Multi-Dimensional Parameter Space Example

This section demonstrates the utility of the TOPE process for multi-dimensional parameter spaces. The experimental technique and TOPE process are the same as used in the previous

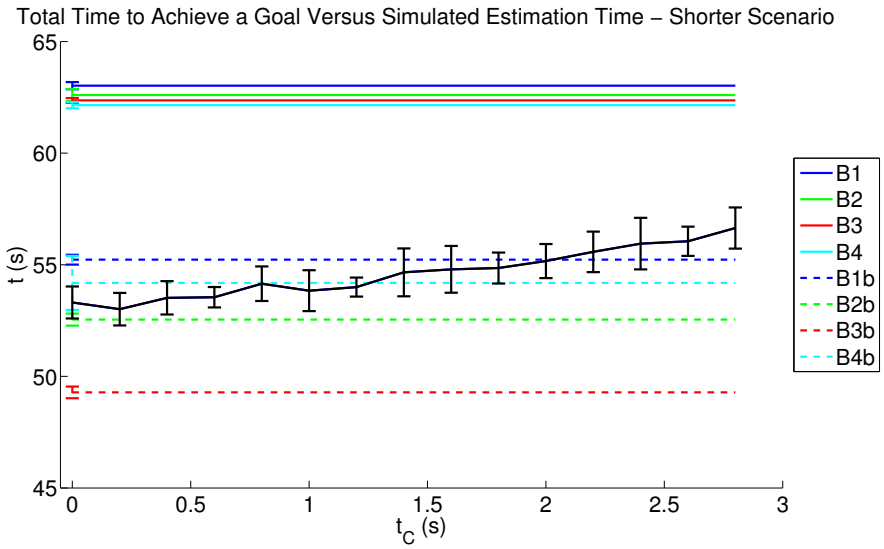


Figure 4.6: Total time required to achieve a goal versus estimation time, for the replanning scenario with a parameter space consisting of a discrete set of cell size values. The coloured lines are the total time for each of the comparison techniques, and are horizontal because they are independent of the TOPE estimation time. The black line is the TOPE performance, which is seen to depend on the estimation time required. The error bars show two standard deviations from the mean after 10 repetitions.

section, but the parameter space is the combination of both the heuristic weighting and state space discretisation spaces. The heuristic weighting parameter space adjusts the inflation factor, ε, for a weighted A^* search, and the specific set of values used is the same as that discussed in Section 3.3, and used in Figures 3.6 and 3.7. The brute-force parameter search is performed over every combination of these two parameters, allowing the estimator to choose the optimum combined parameter set (as was shown in Figure 3.8 for the static state space).

Figure 4.7 shows the results for this combined parameter space. The total time to achieve a goal using the TOPE estimation process is further reduced from the results shown in Figure 4.6, for all simulated estimation times. As a result, the TOPE estimation process is superior to all uninformed static choices of parameters (B1 and B2), but also even the informed techniques (B1b and B2b). Furthermore, if the simulated estimation time is less than $0.6s$, then the TOPE process is superior to all the baseline techniques, including those which benefit from the use of anytime techniques (B3b and B4b).

There is one data point for which the previous statement is untrue. When the simulated

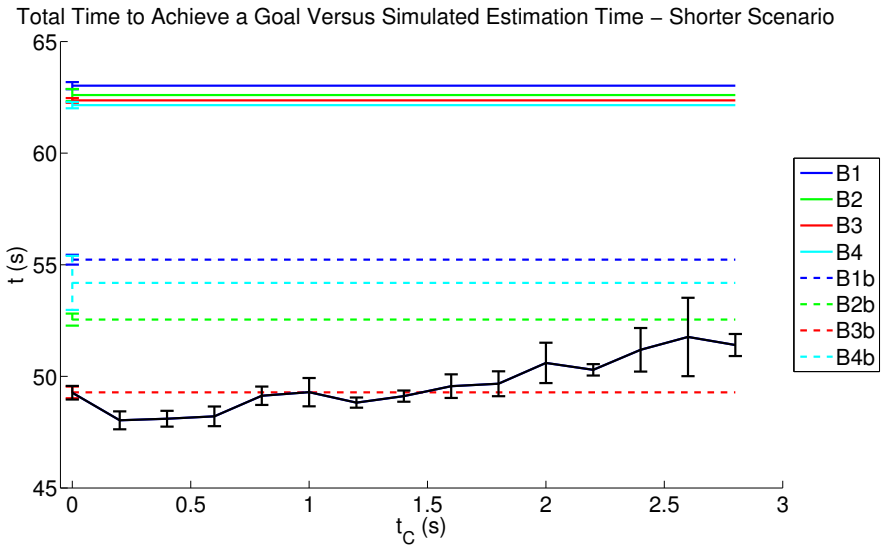


Figure 4.7: Total time required to achieve a goal versus estimation time, for the replanning scenario with a two-dimensional parameter space containing discrete sets of both cell size and ε values. The coloured lines are the total time for each of the comparison techniques, and are horizontal because they are independent of the TOPE estimation time. The black line is the TOPE performance, which is seen to depend on the estimation time required. The error bars show two standard deviations from the mean after 10 repetitions.

estimation time is $0s$ an increase in total time of approximately $1s$ ($< 2\%$ of the total time) is seen. This effect is known as the *switching cost*, and reflects the time required to append a subset of the new plan to the committed plan being executed by the vehicle model. When t_C is very small, new plans can be computed frequently, increasing the total switching cost. In general, this cost reflects the difference between t_E and $\hat{t_E}$; if the agent model and reality are in agreement, such that $t_E = \hat{t_E}$, then the switching cost is negligible.

Once again however, the numerical values of these results are less important than the confirmation that an increased dimensionality of the parameter space allows a wider variety of tuning adjustments to the system, and has the potential to further improve the performance gains of the TOPE process over alternative methods.

4.3.6 Summary

These results illustrate the potential gains from employing a TOPE estimation process, especially when there are multiple parameters that can affect the planning versus execution

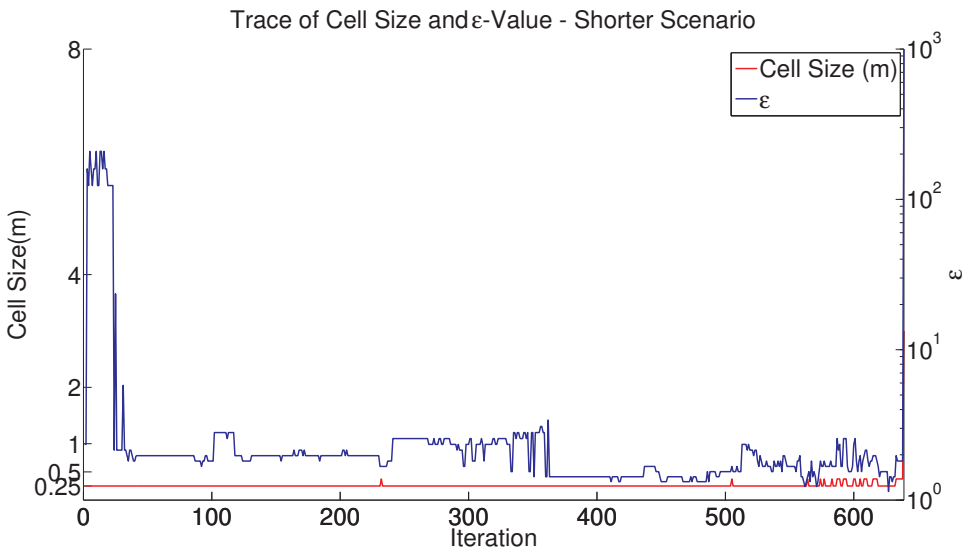


Figure 4.8: Optimum cell size and ε values at each iteration through the TOPE process, showing that no single *a priori* tuning can capture the optimum trade-offs.

time trade-off. Figure 4.8 further reinforces the need for such techniques, by illustrating a *trace* of the optimum cell size and ε values at each iteration during the TOPE process (using the run with $t_C = 0.0s$ from Figure 4.7). This trace shows that the optimum ε values vary across nearly the full range, but that the optimum cell size value is almost always $0.25m$, for this particular scenario and parameter space.

Although it may be possible to select fixed values of some parameters that give close to optimal performance, no single *a priori* tuning can capture the optimum trade-offs in every situation. As an example, for the longer 2D planning scenario (for which it is impractical to perform the brute-force parameter search since the time required is excessive), the optimum cell size will definitely not be $0.25m$ at every iteration. This can be verified with reference to Figure 3.1, which shows that for at least the first iteration this will result in a plan taking greater than $15s$ to compute. Nor will the optimum cell size remain fixed at $2m$ as given in the same figure, since near the end of the scenario when the length of the required plans are less than or equal to the shorter 2D planning scenario, it can be expected that lower cell sizes would be superior.

This section has presented a general procedure for the use of the TOPE process within a replanning system. The experiments confirmed two hypotheses; firstly that the TOPE process can yield improved performance over any system employing fixed parameter values

from the same parameter space, and secondly that performance can be expected to improve further when additional parameters are available to adjust at run time. The results cannot show that the TOPE process will be capable of this for every replanning system or scenario, only that it is true for the scenarios presented above. The results are also only true when the Equation 4.9 can be solved with perfect accuracy. The following section performs a sensitivity analysis of the TOPE process in order to assess whether this requirement can be removed.

4.4 Sensitivity Analysis

The purpose of this section is to analyse whether the TOPE process can be run in practice, given that the previous analysis only showed that it was possible with perfectly accurate solutions to Equation 4.9. Without the use of a brute-force parameter search, it is unlikely that perfect accuracy can be achieved, and yet such a search cannot be used in practice because it requires excessive time to do so. What is required instead is knowledge of the *operating region* for the TOPE process; how fast it must run for a given accuracy, and vice-versa.

Section 4.4.1 presents a procedure to determine the operating region of the TOPE process, by way of a perturbation analysis augmented to the simulation used previously. This is followed by Section 4.4.2 which describes how this procedure is used in two experiments that aim to determine whether a suitable operating region exists for the 2D planning scenario.

The first is described in Section 4.4.3, and has the hypothesis that as the time required for the TOPE process is increased, the accuracy required by the calculation of y^* will also increase. The second experiment is described in Section 4.4.4, and applies the perturbation analysis to the timeliness of the TOPE process, rather than its accuracy. This is also valuable because a real system is unlikely to calculate the result of Equation 4.9 with a constant t_C in each iteration.

The procedure used in both the experiments is a form of sensitivity analysis; assessing the sensitivity of a replanning system to the timeliness and accuracy of the TOPE process. When used in comparison with alternative methods, the procedure can also determine the operating region within which the TOPE process can reduce the total time required to achieve a goal.

4.4.1 General Procedure

The general procedure to perform sensitivity analysis of the performance of a replanning system to the accuracy or timeliness of the TOPE process is shown in Algorithm 4.2. It is based on the procedure presented in Algorithm 4.1, but with the addition of lines 3-8 which perturb y^* or t_C in each iteration, and the requirement after line 8 which ensures that the procedure waits appropriately if a perturbed t_C is greater than the time that was actually required.

Algorithm 4.2 A general procedure for analysis of the sensitivity of a parameter space to perturbations in estimation accuracy or timeliness.

1: **while** goal has not been achieved by the planning system **do**
2: determine y^* using Equation 4.9, taking time t_C
3: **if** assessing TOPE accuracy **then**
4: $y \sim \mathcal{N}(y^*, \delta)$
5: **else if** assessing TOPE timeliness **then**
6: $t_C \sim |\mathcal{N}(t_C, \delta)|$
7: $y \leftarrow y^*$
8: **end if**
Require: t_C has elapsed
9: apply y to planning system
10: compute a new plan from the end of the committed plan to the goal, taking time t_P
11: **if** there are lower levels in the hierarchy **then**
12: send this plan to the next lower level in the hierarchy
13: **else**
14: determine subset of this plan within commitment length
15: send this subset to the agent's controllers as the committed plan
16: **end if**
17: **end while**

The procedure is applicable to multi-dimensional parameter spaces if the perturbation in lines 3-8 allows one or many parameters in the space to be perturbed by the same or different amounts. For use as a design tool however, it is likely to be more useful to vary one parameter at a time, as this is also valuable to assess parameter utility as described in Section 4.2.1.

4.4.2 Experimental Method

This section describes the use of Algorithm 4.2 as a procedure for experiments that aim to determine the available operating region for the TOPE process. The experimental method

is similar to that described in Section 4.3.2, but there are two independent parameters rather than one. The first independent parameter is still t_C, the simulated time required to compute the result of Equation 4.9, and the second affects the amount by which the parameter being perturbed should be adjusted, and is labeled δ.

In each iteration of the procedure the true value of the parameter to be perturbed is found using 4.9 (either the output y^*, or the measured t_C). The perturbed value of this parameter is found by drawing a sample from a Gaussian distribution with mean equal to the true value, and variance equal to δ. In each experiment, δ is varied from zero up to some maximum amount. In the case of parameter value perturbation, this maximum amount should be proportional to the range of values which the parameter(s) can take. In the case of t_C perturbation, δ can simply be any suitable value, but the perturbed value can only be drawn from the positive half of the distribution (or by taking the absolute value of each sample).

There are two implementation details worth discussing. The first occurs when perturbing parameter values from a parameter space with discrete components, such as the state space discretisation used in the previous and following examples. In this case, the perturbed value must also be rounded to the nearest discrete value. The second is the effect of failed plans that can occur when the adjustment process causes the planner to be configured with ineffective tuning parameters. This cannot occur in the original simulation, since y^* is always associated with a valid plan. In this sensitivity assessment however, since y^* is adjusted after the brute-force parameter search, it is possible for the planner to fail when configured with the adjusted y. This is handled in the same manner that the PHR handles global plan failure; simply continuing execution along the committed plan, and pausing execution if no updated plan is available before the agent reaches the end of the commitment extent. Any time spent paused is accrued as part of the total time required to achieve a goal.

For both experiments presented below, the parameter space contains only the state space discretisation parameter. A one-dimensional parameter space was used because the addition of a second dependent parameter increases the time required by the experiment significantly, making analysis of multiple parameter sets impractical. For the same reasons, the experiments are performed only for the shorter 2D planning scenario.

For each experiment, two pairs of figures illustrate the results. In each pair, the first figure

shows the mean total time required to achieve a goal versus the time required to compute each instance of Equation 4.9, and the amount of adjustment made to either y^* or t_C. The colour indicates the size of two standard deviations from the mean at each point, given the scale on the right of the figure. Also shown is a plane which is the result of the comparison method B1b. As above, B1b is the appropriate comparison because it operates in the same parameter space (cell size only) and uses the best performing static parameter values.

To use these results as a design tool, the second figure in each pair shows the upper 95% confidence bound on the mean total time, compared with the lower 95% confidence bound on the comparison result. In this figure, any parts of the coloured surface (where colour is proportional to total time) that lie below the lower bound on B1b's performance, are operating regions for the TOPE process that will result in a lower total time to achieve a goal compared to B1b, with a 95% confidence. In other words, these are the regions of the first figure for which the upper and lower 2σ error bounds are separated for the TOPE and comparison processes, respectively.

4.4.3 Sensitivity to Estimation Accuracy

This section presents the results of the first sensitivity experiment, which perturbs the values of y^* determined by the TOPE process in each iteration. The hypothesis of this experiment is that as the time required for the TOPE process is increased, the accuracy required by the calculation of y^* will also increase.

Figure 4.9 shows the surface representing the mean total time to achieve a goal as t_C is varied from $0s$ to $3s$, and the value of two standard deviations of the perturbation distribution is varied from 0% to 20% of the full range of cell size values ($0.25m$ to $8.0m$). The surface is seen to slope upwards (indicating a longer total time) as both the standard deviation of the perturbations and t_C are increased. The former slope confirms the hypothesis of this experiment, and the latter reconfirms the result obtained in Section 4.3.

Figure 4.10 shows the operating region for the TOPE process as compared to the B1b technique, for variation in the accuracy with which Equation 4.9 can be calculated. This operating region is the small part of the surface that lies below the grey plane. TOPE processes with accuracy and timeliness within the bounds of this region can be expected to improve upon the performance of the B1b technique with 95% confidence. This region

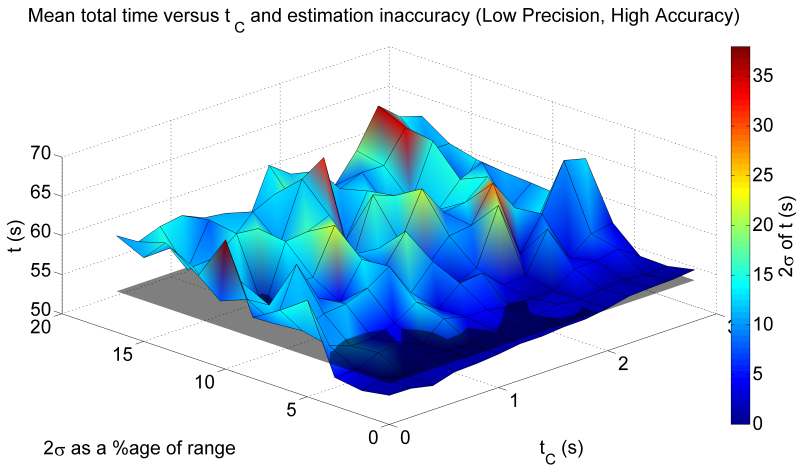


Figure 4.9: Example sensitivity analysis of TOPE process accuracy.

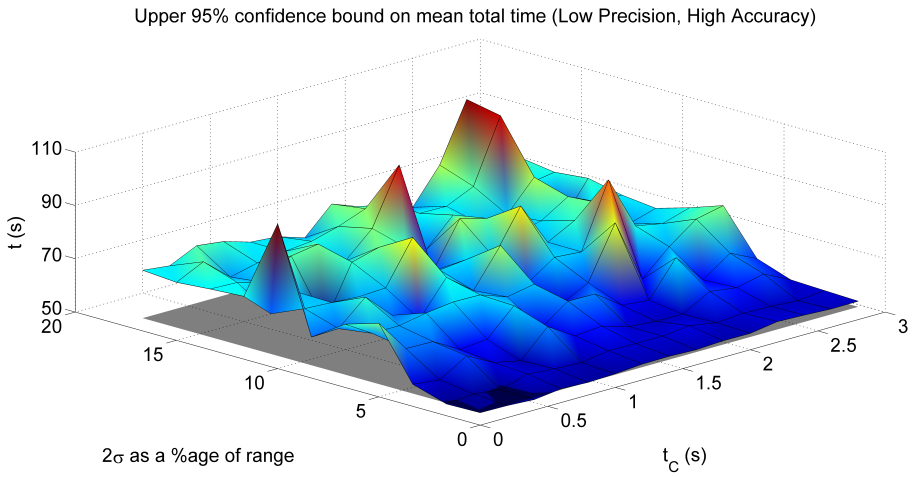


Figure 4.10: Operating region of the TOPE process subject to inaccuracies in the calculation of Equation 4.9.

is very small, requiring $t_C \leq 0.4s$ and the standard deviation of cell size values determined by Equation 4.9 to be kept within 1% of the $8m$ range from the optimum value (implying that 95% of the results are kept within ±2% of the true optimum solution).

Although this range is very small for this particular scenario and parameter space, it is important to note that the B1b comparison technique is the best performing technique possible using only fixed parameter values. Any other technique using fixed parameter values will have worse performance than B1b, and thus the operating region of the TOPE process as compared to other techniques will be larger than that shown. Since Figure 4.8 indicated that the optimum cell size value in any iteration was approximately constant for this scenario, this suggests that fixed parameter values are likely to be more effective in a scenario such as this than in any other. It is reasonable to expect then that the operating region of the TOPE process by comparison with B1b may be larger in other scenarios.

4.4.4 Sensitivity to Non-Constant Estimation Time

This section presents the results of the second sensitivity experiment, which perturbs the values of t_C taken by the TOPE process in each iteration, but leaves y^* as is. The hypothesis of this experiment is that as the variance of the distribution of t_C increases, the time required for the TOPE process to achieve a goal will increase. In addition, it is expected that the variance on total time will also increase, since the time required by an individual iteration of the process can be higher than it would otherwise be.

The following pair of graphs again shows the mean total time to achieve a goal in Figure 4.11, and the upper 95% confidence bound on the mean total time, compared with the lower 95% confidence bound on the comparison B1b result in Figure 4.12.

Figure 4.11 shows the surface representing the mean total time to achieve a goal as t_C is varied from $0s$ to $3s$, and the value of two standard deviations of the perturbation distribution is also varied from $0s$ to $3s$. The surface is seen to slope upwards (indicating a longer total time) as both the standard deviation of the perturbations is increased, and as t_C is increased. The former slope confirms the first hypothesis of this experiment, and the latter reconfirms the result obtained in Section 4.3. The colour scale represents the value of two standard deviations of the total time at each point on the surface. This scale is used to confirm the second hypothesis, by the observation that as the variance of the distribution

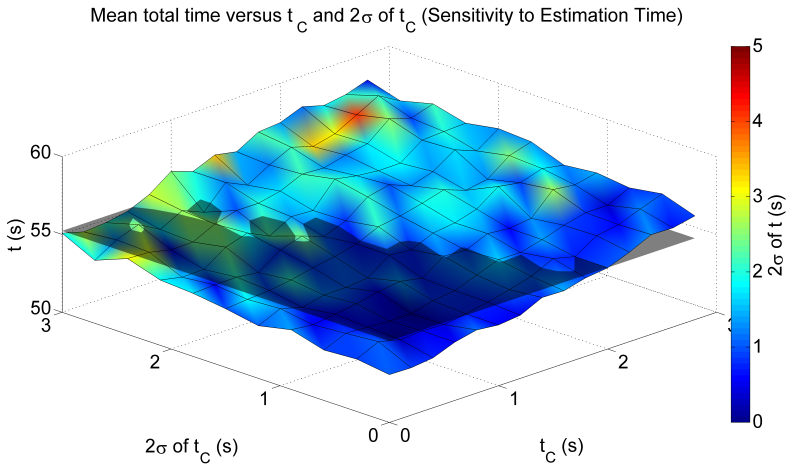


Figure 4.11: Example sensitivity analysis of TOPE process timeliness.

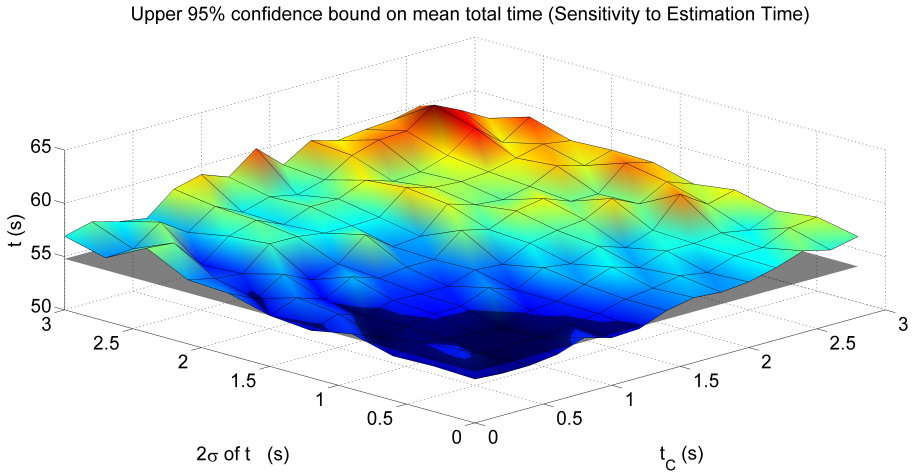


Figure 4.12: Operating region of the TOPE process subject to inaccuracies in the time required to calculate of Equation 4.9.

of t_C increases, the variance in total time values tends to also increase.

Figure 4.12 again shows the operating region for the TOPE process as compared to the B1b technique, but for variation in the time to calculate the result of Equation 4.9. The region appears larger[4] than that of the previous experiment, with the distribution of t_C able to have $2\sigma \leq 1s$ if the mean value of t_C is below $0.4s$. If the mean value of t_C increases to $1s$ then the distribution can tolerate $2\sigma \leq 0.4s$ and still result in a lower total time than the B1b technique.

As for the previous experiment, these results and operating region are only valid for the particular scenario and parameter space being used. For the shorter 2D planning scenario and cell size parameter space, the combination of results from Figures 4.9 to 4.12 suggest that there exists an operating region within which the TOPE process need not be instantaneous or perfectly accurate, but will still have better performance than any other technique using only fixed parameter values.

4.4.5 Summary

This section has presented a general procedure for assessing the sensitivity of the TOPE process to variation in both the accuracy and timeliness of calculation of Equation 4.9. Experiments were performed that ran this procedure in simulated time using a brute-force parameter search to determine accurate results in fixed time periods. These results and time periods were then perturbed by taking random samples drawn from distributions centred on the correct results, and with increasing variance. These results were then displayed as a surface and compared with the plane corresponding to an alternative technique employing fixed parameter values.

For the shorter 2D planning scenario and cell size parameter space, it was shown that tight but usable operating regions exist within which the TOPE process can result in a reduced total time to achieve a goal than the comparison technique. This comparison technique was B1b, which represents the best possible selection of fixed parameter values for the cell size parameter space. It can thus be said that the TOPE process is capable of better

[4]This suggests that a perturbation on t_C of $2\sigma = 3s$ is of less significance than a perturbation on y^* of $2\sigma = 20\%$ of $8m$, for the cell size parameter space and shorter 2D planning scenario. However, it does not necessarily suggest that sensitivity to the time required to calculate the results of Equation 4.9 will always be less than sensitivity to the accuracy of these results.

performance than any technique using only fixed parameter values, even when it is neither instantaneous nor perfectly accurate.

As with other experiments within this chapter, the results are only valid for this particular scenario and parameter space, but the procedure is generally applicable. In addition, the technique can be used without a brute-force parameter search or simulated time. This means that it can be used in any system which is capable of replaying state space data and simulating the operation of an agent and its replanning system. The sensitivity analysis procedure thus provides a measure of parameter utility, since it can be used to compare the sensitivity of the system to perturbations in any of its tuning parameters.

4.5 Conclusion

This chapter defined the TOPE problem, and derived the equations that describe it as a problem of continually determining the sets of system parameters that are expected to minimise the total time required to achieve a goal. Subsequent sections described the parameter space from which these sets are drawn, demonstrated that this parameter space can include the choice of which parameters are most useful, and presented several qualitative (but not quantitative) measures of parameter utility to assess this.

The TOPE process was then presented as a means of solving the TOPE problem within a replanning system. The description of this process is one of the main contributions of this thesis, and it was suggested that if the process was performed sufficiently quickly and accurately, it had the potential to outperform any fixed parameter technique. This was shown to be true for at least the example scenario, where the TOPE process was able to outperform the best possible fixed parameter technique, implying it would outperform *any* fixed parameter technique in that scenario.

A procedure to perform sensitivity analysis of the TOPE process was then presented, and demonstrated on the example scenario. For this example, it was shown that the sensitivity of the TOPE process to the accuracy and timeliness of the calculation of Equation 4.9 was sufficiently low that there existed a viable operating region. If a real TOPE estimator could be designed with accuracy and timeliness within this operating region, it could be expected to outperform the best possible fixed parameter techniques for the example scenario, at a 95% confidence level. It was further shown that any other fixed parameter technique would

have worse performance, leading to an even larger operating region in which the TOPE process would outperform this technique.

There is one important caveat to all the experiments presented in this chapter; that no real-time implementation of a TOPE process was required or used, since the experiments used the optimum parameter sets obtained by a brute-force parameter search as a form of ground-truth information. As a result of this, t_C was a constant in each iteration of the TOPE process, since it was an independent variable for each of the experiments. The exception to this was the analysis of the sensitivity to variations in the time required to calculate Equation 4.9, in which the recorded t_C was adjusted, and shown to increase the total time required to achieve a goal as the amount of variation was increased.

Chapter 5 shows that real-time implementations of the TOPE process require a form of estimation, in order to find the results of Equation 4.9 in a manner that is sufficiently fast and accurate. The chapter presents the two main classes of estimation techniques suitable for the TOPE process, and describes the design and implementation of examples of each. These examples are compared with existing techniques as a baseline, showing that the process is indeed suitable for a practical and realistic planning problem.

Chapter 5

Implementation Results and Analysis

In the previous chapter, a brute-force parameter search was used to determine the most beneficial parameter set at each iteration of the TOPE process. The results of this approach provided a ground-truth baseline, from which it was shown that a perfectly accurate TOPE process is capable of better performance than any other method using only fixed parameter values. Perturbing the ground-truth results found by this brute-force parameter search further showed that an inaccurate TOPE process is also capable of better performance than other methods, provided that the process is sufficiently fast.

However, such a brute-force parameter search is completely unsuitable for real-time operation. By way of example, the brute-force parameter search of the combined heuristic weighting and state space discretisation parameter space requires just over two minutes of computation time in the first TOPE iteration (when the start to goal distance, and thus the planning time, is highest). In total, the brute-force approach required nearly ten hours of computation to perform the several thousand plans necessary for the shorter 2D planning scenario shown in Figure 2.2, despite the total execution time being just under 50 seconds.

The aim of this chapter is to demonstrate practical estimation techniques to solve Equation 4.9 in a sufficiently timely and accurate manner, such that the TOPE process can improve upon the performance of existing methods. It is also shown that these existing methods, such as anytime algorithms, can be described as particular estimator implementations within the

TOPE framework. The implementations of estimation techniques as applied to the TOPE process are referred throughout this chapter as *TOPE estimators*.

The TOPE estimators described in this chapter are separated into two main classes; heuristic and statistical. Similar to the distinction between reactive and deliberative methods in the planning domain, this separation is conditioned on whether the estimators make a choice of parameters given only a few or no prior results, or whether they record all prior results and make a decision given some function of the entire history. Statistical estimators are further subdivided into those which actually compute plans to determine the exact result of a choice of parameter set, and those which use a second statistical model to predict the result of parameter set choices without computing each plan.

The structure of this chapter is as follows. Section 5.1 describes each of the classes of estimation technique. Section 5.2 then describes the implementations of several examples of each category, in addition to comparison techniques. Section 5.3 presents the results of each category of techniques and compares them within categories. Section **??** discusses the results across categories, and describes the implications for the TOPE process. Finally, Section 5.5 concludes the chapter with a discussion of the implications of these results with reference to the original thesis statement.

5.1 Classes of TOPE Estimation Techniques

This section describes TOPE estimation techniques suitable for real-time implementation, and separates them into two classes; heuristic and statistical. In either case, the purpose of the estimation is to find the set of parameters that minimises the expected total time to achieve a goal, using Equation 4.9. As such, the estimation process is an optimisation of a metric function:

$$m = \hat{t_C} + \hat{t_P} + \hat{t_E} \tag{5.1}$$

applied at each point in the parameter space (which may be discrete or continuous for each parameter). The resulting surface is henceforth referred to as a *metric surface*.

It should be noted that if the parameter space does not include any factors which affect t_C, then t_C is constant over the entire surface and can be removed from the metric function, leaving $m = \hat{t_P} + \hat{t_E}$. This is the case for the examples presented in this chapter, since

their parameter space is the combination of the heuristic inflation factor and grid cell size. Although t_C affects the total time (and remains important to minimise), the choice of parameters does not affect t_C itself.

The brute-force parameter search used in Chapter 4 was able to assess this metric function at each point in the parameter space to create the full metric surface at each iteration. This is impractical for real-time operation, since the time required to determine the surface is excessive. The brute-force parameter search was also able to actually compute each plan, giving it a highly accurate measure of t_P, and the best possible estimate of $\hat{t_E}$. For real-time operation, this would mean the time taken to perform a single sample of the metric surface would be equal to the planning time of the plan with the sample's parameters. It is thus unlikely that an estimation technique could make multiple samples using this assessment technique, and still have a sufficiently low t_C[1].

To further complicate matters, the metric surface is also non-stationary with respect to time. As the planning scenario proceeds, the metric value associated with any parameter set can be expected to decrease by some amount related to the amount of execution of prior plans that has occurred. However, since the planning state space is dynamic, a previous metric value associated with any parameter set could be completely inaccurate with or without any intervening execution.

Figure 5.1 is a reprint of Figure 3.9 and shows an example of this metric surface for the combined parameter space of heuristic weighting and grid cell size, in the shorter 2D planning scenario shown in Figure 2.2. A video available from `http://www-personal.acfr.usyd.edu.au/t.allen/PhdThesis/AnimationOfTopeEstimatorMetricSurface.avi` shows an animation of this metric surface at each iteration of the TOPE process. The data was taken from a run of the experiment described in Section 4.3.5 with $t_C = 0.0s$, and the optimum cell size and ε reported in each iteration are those shown in Figure 4.8.

The following two sections describe heuristic and statistical estimation techniques that determine approximations to the true optimum parameter sets, despite the difficulties described above. In simplified terms, the heuristic techniques avoid the time intensive sampling of the metric surface by relying on domain specific knowledge to approximate parameter

[1]Despite this, some of the estimation techniques described later in this chapter do perform multiple samples, however they store the output plans of each sample so that planning is not actually required. This has the effect of reducing t_P to zero, however the total $t_C + t_P$ remains high.

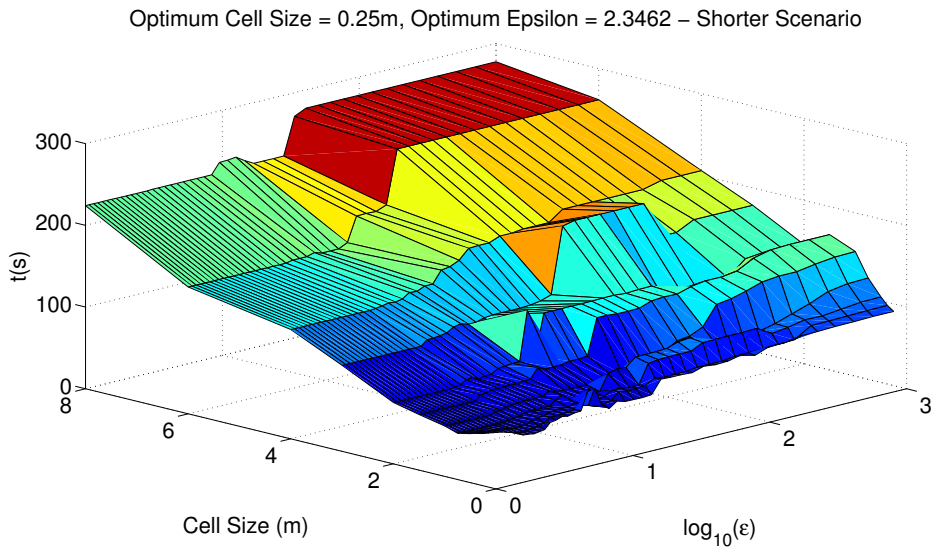


Figure 5.1: Total time required to achieve a goal in the shorter static planning scenario versus the choice of heuristic weighting and grid cell size, using the A^* algorithm (reprinted from Figure 3.9).

sets that are sufficient for the particular scenario. Given this specific knowledge however, they are unlikely to be generally applicable. The statistical techniques typically perform limited sampling in each iteration and assume the metric surface is approximately stationary with respect to time. Alternatively, they approximate the results of each sample rather than actually performing a plan with the sample parameters, such that they are able to perform large numbers of samples in each iteration of the estimation process.

5.1.1 Heuristic Techniques

Heuristic estimation techniques employ domain knowledge to select parameters that are expected to be sufficient for the application. Although the selected parameters are unlikely to be the optimum, they are typically able to be determined very fast (often in constant time), which can make up for their inaccuracy (as was demonstrated by the sensitivity analysis in Section 4.4.3).

The use of domain-specific heuristics to alter parameters during operation is not new. For example, Ferguson et al. (2005) suggests that many systems using the D^* *Lite* algorithm "abort the replanning process and restart from scratch when either major edge cost changes

are detected or some predefined threshold of replanning processing is reached." Similarly, Likhachev & Ferguson (2009) uses an anytime approach and "if map updates are significant [...] then ε is set back to its initial value." Both of these techniques encode domain knowledge for these particular planning algorithms, and could be incorporated into an estimator as a set of condition-action rules.

Anytime algorithms can also be considered a heuristic estimation technique, where ε is the only parameter, and the heuristic suggests that reducing this parameter by some amount in each iteration is likely to reduce the total time. For both anytime algorithms and the above examples however, there is no feedback to determine whether the parameter adjustment was effective.

While most heuristic techniques employ domain specific knowledge, there are some heuristics that are generally applicable across TOPE implementations. Two such heuristics of particular value are as follows:

1. On the first iteration of the TOPE process, execution cannot begin until a plan is computed. As a result, a rapidly computed plan which initiates execution in an approximately correct manner is typically beneficial for reducing the total time required to achieve a goal. Thus, selecting the parameter set that minimises $\hat{t_C} + \hat{t_P}$ (rather than $\hat{t_C} + \hat{t_P} + \hat{t_E}$) on the first iteration is often a practical heuristic.

2. Since the TOPE process is intended to be used in a hierarchical planning system such as the PHR, there is always the option to continue execution on the previous plan, and typically with the safety of the agent ensured by a lower level planner. Thus, it is possible to abort a plan if it exceeds a maximum time-limit, with execution continuing on the previous plan. In most cases, aborting a plan that requires excessive computation can be more beneficial than continuing to plan, especially if the state space information has changed extensively during the planning process.

The first of these heuristics can be retro-fitted to almost any system with run-time adjustable parameters, as was done to the PHR system to yield a significant performance improvement (Allen et al. 2009). The second can be used as stated, or as part of a feedback mechanism whereby any plan which exceeds the time limit is considered to be the result of a poor parameter adjustment decision. Some of the heuristic estimator implementations described

in Section 5.2 use this technique to revert poor decisions, resulting in a form of gradient descent of the metric surface.

The categorisation of heuristic estimation techniques is defined by the fact that they make use of a few or no prior results. Of the techniques described above, most make no use of prior results at all, instead relying only upon domain knowledge and assumptions. Those described as using a form of feedback are making use of the immediately prior result (or two prior) only, and assessing whether the decision made in that iteration (or between those two iterations) was effective.

5.1.2 Statistical Techniques

By contrast with heuristic estimation techniques which make decisions on the basis of a few or no prior results, statistical techniques record many prior results in order to build a model of the metric surface. Given that the metric surface has the potential to change unpredictably, and that the starting point of each plan changes between iterations, this model is necessarily imperfect. It is only effective if the metric surface can be assumed not to change significantly between iterations, and it cannot be used for memoization[2] of plans themselves, but to record their metric value only.

The means by which this model is built subdivides statistical estimators into two sub-categories. The first is that in which the estimators sample points on the metric surface by directly computing the plan with the parameter values of the sample, and then record the metric value of each plan. Since the time taken to perform each sample is at least equal to the planning time required, this limits the number of samples that can be performed in each estimation iteration without t_C reaching impractical levels.

The second sub-category is that in which the estimators use a model that maps state variables and parameter sets to metric values. This model can be hard-coded, learnt by supervisory techniques or autonomous techniques (Karumanchi et al. 2009), or built incrementally as the TOPE process continues. While the metric values suggested by such models are unlikely to prove as accurate as those found by actually computing the plan,

[2]Memoization refers to recording the output of a function given a particular input, such that this output can be rapidly retrieved when the same input is used again, rather than recalculating the result. With reference to plans, this would imply that for a given parameter set, the entire plan could be stored and reused if the same parameters were selected again. Since the agent's state will differ between TOPE estimation iterations, as will the planning state space, this is not possible within the TOPE process.

this sub-category avoids the need to assume that the metric surface is approximately stationary with respect to time. An example model for a 2D path planning scenario using a graph search algorithm could use the fact that the distance between the start and goal is correlated with planning time, since it affects the number of states that must be expanded by the algorithm. By collecting statistics of this relationship, a statistical estimator could map from path length to the parameters that are expected to minimise the total time to achieve a goal. In reality, the learnt models may need to assess many more state variables than this example to be sufficiently accurate, since planning time is also affected by many other factors.

The biggest drawback to the second sub-category is that the models used are inherently application specific, whereas the framework of the TOPE process is general. The effort required to hard-code a model, supervise a learning process, or implement an incremental approach is undesirable if the technique is not readily portable to a subsequent application. For this reason, the implementations described in Section 5.2 focus on the first sub-category, and the remainder of this section details techniques to find the parameters at the global minimum of the metric surface using this approach.

As shown by Figure 5.1, the surface may have many local minima, meaning that gradient descent approaches are unlikely to prove effective in all situations. Given that the act of taking an accurate sample on the metric surface requires performing the plan and accruing this planning time, it is likely to be impractical to use techniques which require an undefined number of samples. This means that techniques such as Expectation Maximisation (EM) are unlikely to be effective, since they continue making samples until the result converges to some bound (meaning that additional samples are unlikely to affect the current estimate of the optimum metric value and parameters).

Techniques from the fields of Reinforcement Learning (Sutton & Barto 1998) and Monte-Carlo Methods (Hammersley & Handscomb 1975) can provide a practical compromise between the desire to have accurate samples of the metric surface, as well as needing to perform each estimation iteration with a sufficiently low t_C. The general procedure of such techniques is to combine greedy exploitation of prior results, with exploratory learning of new samples. In the TOPE framework this corresponds to sometimes using the current best known parameter set, and sometimes learning the result of a random parameter set.

Many such methods, including Temporal-Difference Learning (TDL) (Tesauro 1995, Sutton

& Barto 1998), Q-Learning (Watkins 1989), and Simulated Annealing (SA) (Kirkpatrick et al. 1983, Cerny 1985), use a 'learning-discount parameter'[3], $p \in [0, 1]$, where p decreases from p_{max} to p_{min} by some Δp in each iteration. The value of p in any iteration is the probability that a random sample will be used, and the corresponding value of $1 - p$ is the probability that exploitation of the current best known sample will be used. Some of the statistical estimator implementations described in Section 5.2 use these techniques, however the best performing variation performs plans using both the current best and a random parameter set at each iteration, and chooses the better result.

5.2 TOPE Estimator Implementations

This section describes the implementation of several heuristic and statistical estimators for the 2D planning scenarios shown in Figures 2.1 and 2.2, using a two-dimensional parameter set comprising of the heuristic weighting and state space discretisation parameters. The results of each implemented technique are compared with several baseline results, with and without the use of *a priori* information about the state space. In each case, the results are obtained from the same CORD vehicle simulation used for the analysis in Chapter 4, save that the estimates of the appropriate heuristic weighting and state space discretisation parameters are determined by a unique estimator implementation for each technique, rather than the brute-force parameter search method. Each TOPE estimator runs in real-time and measures its own runtime as the reported t_C, rather than simulating this as per the experiments described in Chapter 4.

Each technique is labelled according to its category: Baseline (B); Heuristic (H); or Statistical (S), and a number is appended to distinguish different techniques within each category. In all the following descriptions, c refers to the side length of a square grid cell, and ε is the inflation factor applied to the heuristic used by the global planning algorithm.

The number of possible implementations in each of the heuristic and statistical categories is enormous, and in these experiments, only four examples of each type are demonstrated. The example estimators were chosen by selecting a method or methods that appeared promising, and implementing subsequent examples by extending the most successful concepts in each case. As a result, within each category a higher label number typically indicates a more

[3]The learning-discount parameter is functionally equivalent to the 'temperature' used in SA.

		Additional Information		No Additional Information	
	Parameters:	c	ε	c	ε
Estimator	B1	0.375	1.0	1.0	1.0
	B2	0.25	2.3462	1.0	1.5
	B3	0.375	variable	1.0	variable
	B4	0. 375	variable	1.0	variable

Figure 5.2: Cell size and heuristic weighting parameter values for the two categories of baseline estimation techniques in the shorter scenario.

sophisticated technique. The effect of implementing the example estimators in this manner is that the results are indicative only of the performance of these examples as compared to the baseline techniques. It is likely that both better and worse performing implementations are possible for this parameter space.

5.2.1 Baseline Techniques

The baseline category contains the eight techniques presented in Section 4.3.3, comprising four main techniques with two sets of parameter values each.

- B1 uses the A^* algorithm with a fixed c and $\varepsilon = 1$.
- B2 uses the Weighted A^* algorithm with a fixed c and a fixed $\varepsilon > 1$.
- B3 uses the Anytime-A^* algorithm with a fixed c.
- B4 uses the Anytime-D^* algorithm with a fixed c.

The differences in c and ε values between the two categories of baseline techniques are given in the tables shown in Figure 5.2 for the shorter scenario, and in Figure 5.3 for the longer scenario. As per Section 4.3.3, those techniques with additional information have a lowercase 'b' appended to their label.

5.2.2 Heuristic Implementations

Since heuristic estimation techniques draw upon domain specific knowledge to determine the appropriate parameter sets, this section first describes relevant knowledge from the

		Additional Information		No Additional Information	
	Parameters:	c	ε	c	ε
Estimator	B1	2.0	1.0	1.0	1.0
	B2	0.75	1.9745	1.0	1.5
	B3	2.0	variable	1.0	variable
	B4	2.0	variable	1.0	variable

Figure 5.3: Cell size and heuristic weighting parameter values for the two categories of baseline estimation techniques in the longer scenario.

2D planning scenario. This knowledge is obtained from the experiments and simulations shown in Chapters 3 and 4, and several heuristics are derived to exploit it. In addition to domain specific knowledge, these heuristic estimators also exploit the generally applicable heuristics described in Section 5.1.1. Each of the estimators then combines one or more of these heuristics to determine appropriate parameter sets at each estimation step.

The following list provides a selection of relevant items of domain knowledge for the 2D planning scenario, with a parameter space that incorporates both the cell size and heuristic inflation factors:

- From Figure 4.8 it is seen that ε^* is never found to be the minimum value of 1.0, and is very rarely below a value of approximately 1.15.
- Also from Figure 4.8 it is suggested that the TOPE process is more tolerant of underestimates of cell size than overestimates.
- From analysis of the brute-force parameter search results, if a plan with a particular ε exceeds a given planning time, then any plan in the same state space with a lower ε must expand at least the same number of states, and will take at least the same amount of time[4]. This suggests that if the estimator is to make a choice between two possible values of ε, it should assess the higher value first, and skip the lower value if the first plan exceeded the time limit.
- From Figure 4.7, if t_C is too high, then there is a definite deterioration in performance regardless of the accuracy of the estimation. Thus if t_C exceeds some $t_{C_{max}}$ the esti-

[4]The same is not necessarily true for the cell size parameter, since larger cell size representations may block otherwise accessible plans in the space.

mation step could be aborted, either allowing execution to continue along the previous plan, or allowing planning to proceed with the previous parameter set. A practical compromise is to combine planning within the estimator such that it computes one plan (with the current best or some default parameters) and then continues with its strategy, but falls back to this plan if the time-limit is exceeded.

Using these heuristics, and the generally applicable additions described in Section 5.1.1, the following list describes several implementations of heuristic TOPE estimators. In the following descriptions, iterations are numbered with the counter, i, and s is a state variable used by H3 and H4 to determine which parameter is to be decreased in a given iteration. For the techniques in which a maximum planning time limit is enforced, its value is $t_{P_{max}} = 0.1s$, which reflects the minimum $10Hz$ replanning rate that was deemed to be sufficient reactivity in prior work on the CORD vehicles when using the PHR system.

- H1 uses the A^* algorithm with fixed c and fixed $\varepsilon = 1.0$. For $i = 1$ only, $c = 8.0m$ and $\varepsilon = 1000.0$, making use of the heuristic that a rapidly computed but suboptimal first plan can be effective.

- H2 uses the A^* algorithm with fixed ε and variable c. For $i = 1$ only, $c = 8.0m$ and $\varepsilon = 1000.0$. For all other i, if $t_{P_{i-1}} \leq t_{P_{max}}$, c is reduced to the next lower value in the discrete set of cell size choices.

- H3 uses the A^* algorithm with variable ε and c. For $i = 1$ only, $c = 8.0m$ and $\varepsilon = 1000.0$. For all other i, if $t_{P_{i-1}} \leq t_{P_{max}}$, either c or ε is reduced to the next lower value in their respective discrete sets of parameter choices. A state variable $s = \{true, false\}$ is defined, and used to determine which parameter is reduced. Starting with $s = true$, c is reduced until $t_{P_{i-1}} > t_{P_{max}}$, at which point $s \rightarrow false$ and c is increased to the next higher value in the discrete set of cell size choices. When $s = false$, ε is selected to be reduced, rather than c.

- H4 is equivalent to H3 except that the condition that reverts a parameter back to its previous choice and switches the state variable s, is an assessment of the difference in the total $t = t_C + t_P + \hat{t_E}$ between the two previous iterations. Defining $t_{E_{i-2 \rightarrow i-1}}$ as the execution time between iterations $i-2$ and $i-1$, if $t_{i-2} - t_{E_{i-2 \rightarrow i-1}} \leq t_{i-1}$, then the decision in $i-1$ increased the expected total time to achieve a goal, and is reverted in the current iteration i, in addition to switching the state variable s.

Like the baseline techniques described above, heuristic techniques can also be augmented or adjusted if additional information is available. Each of the techniques described above is implemented in both an uninformed and informed case, and compared with the appropriate set of baseline techniques. The differences in c and ε values between the two categories of heuristic techniques are given in the tables shown in Figure 5.4 for the shorter scenario, and in Figure 5.5 for the longer scenario. As with the values used in each of the categories of baseline techniques, the informed values are drawn from the analysis and figures shown in Chapter 3, and the uninformed values were determined empirically in the HR and PHR systems' field experiments.

5.2.3 Statistical Implementations

The following list describes several implementations of statistical estimators, using the techniques described in Section 5.1.2. In the following descriptions, iterations are numbered with the counter, i, and p is the probability with which a random parameter set will be used as a sample.

- S1 uses a variation of a TDL technique. With probability p, this approach selects a random c and ε in a given iteration. Alternatively (with probability $1-p$) it selects the best known combination of c and ε determined in previous iterations. The criteria used to determine the best combination of parameters is the total $t = t_C + t_P + \hat{t_E}$ recorded in the iteration they were last used. When a combination of parameters is re-selected, the newer performance statistics (t_P, t_C, etc) replace the previous set, meaning that a set of parameters that were effective in one state space will not necessarily be used if they prove to be poorly performing once the space changes. Like in TDL or SA, the value of p is decreased from p_{max} to p_{min} by subtracting Δp in each iteration. The parameters used were $p_{max} = 1.0$, $p_{min} = 0.1$, and $\Delta p = 0.01$.

- S2 is similar to S1 except that a maximum planning time limit is enforced. If $t_{P_i} > t_{P_{max}}$, planning is aborted, and execution continues using the previous successful plan. The parameters used were $p_{max} = 1.0$, $p_{min} = 0.1$, $\Delta p = 0.01$, and $t_{P_{max}} = 0.1s$.

- S3 is similar to S2 except that the planning and estimation steps are combined into the estimator. In each iteration two plans are performed by the estimator. The first plan

			Additional Information		No Additional Information	
	Parameters:		c	ε	c	ε
Estimator	H1	i=1	8.0	1000.0	8.0	1000.0
		i>1	0.375	1.0	1.0	1.0
	H2	i=1	8.0	1000.0	8.0	1000.0
		i>1	variable	1.5462	variable	1.0
	H3	i=1	8.0	1000.0	8.0	1000.0
		i=2	0.25	2.3462	variable	variable
		i>2	variable	variable	variable	variable
	H4	i=1	8.0	1000.0	8.0	1000.0
		i=2	0.25	2.3462	variable	variable
		i>2	variable	variable	variable	variable

Figure 5.4: Cell size and heuristic weighting parameter values for the two categories of heuristic estimation techniques in the shorter scenario.

			Additional Information		No Additional Information	
	Parameters:		c	ε	c	ε
Estimator	H1	i=1	8.0	1000.0	8.0	1000.0
		i>1	2.0	1.0	1.0	1.0
	H2	i=1	8.0	1000.0	8.0	1000.0
		i>1	variable	1.8167	variable	1.0
	H3	i=1	8.0	1000.0	8.0	1000.0
		i=2	0.75	1.9745	variable	variable
		i>2	variable	variable	variable	variable
	H4	i=1	8.0	1000.0	8.0	1000.0
		i=2	0.75	1.9745	variable	variable
		i>2	variable	variable	variable	variable

Figure 5.5: Cell size and heuristic weighting parameter values for the two categories of heuristic estimation techniques in the longer scenario.

uses a randomly selected combination of parameters, and the second uses the current best known parameter set. The results of both plans are stored in the results database, however only the better performing option is executed. Thus, new parameter sets are still learnt from random samples of the metric surface, but are only used if they are beneficial. The estimator also enforces a maximum planning time, but only for the random samples and not for the best known parameter set. This is done in order that a random sample cannot increase t_C excessively, but the current best parameters can do so as they may still be the best even if the state space changes result in a longer planning time than previously required. As a result, t_P is recorded as the planning time for the better performing option, and t_C is recorded as the total estimation time minus this t_P.

- S4 is similar to S3 except that three options are assessed in each iteration: firstly, the current best parameters; secondly, the current best but deflated according to a heuristic strategy from Section 5.2.2; and thirdly, random selections. The best choice of all three is used, and the estimation time is recorded as the sum of the two unselected planning times. Again, only the random samples have a planning time limit enforced. The heuristic strategy used is H4, meaning that only one parameter is adjusted in each step, but the choice switches whenever the adjustment is ineffective.

5.3 Analysis of TOPE Estimators in Example Scenarios

This section presents the results of running each category of estimation techniques, for both the shorter and longer 2D planning scenarios. The results are displayed on graphs where the horizontal axis labels each technique, and the vertical axis indicates the total time required to achieve a goal. Each scenario within a category of techniques is shown on the same vertical range, but different categories are shown on the range that best illustrates the results. Inter-category comparisons are discussed in Section **??**, where Figures 5.15 and 5.16 show all the results from each category on the one graph, for the shorter and longer scenarios respectively.

The error bars on each data point show $\pm 2\sigma$, which represents the 95% confidence interval after 25 trials of each technique. Techniques were considered to be unsuccessful if they resulted in a total time of greater than $120s$ for the shorter scenario, or greater than $900s$

for the longer scenario, at which point the run was aborted. No techniques that had unsuccessful results had any successful results within their 25 trials.

For all but the baseline techniques themselves, horizontal lines (with $\pm 2\sigma$ error bars shown only on their leftmost end) show the mean values of the relevant subset of baseline comparison results.

5.3.1 Baseline Results

Shorter Scenario

Figure 5.6 shows the mean total time to achieve a goal for each of the four baseline techniques in the shorter scenario, for the case without any prior static analysis. Figure 5.7 shows the same information, but for the case where the information from a static analysis was made available.

Longer Scenario

Figure 5.8 shows the mean total time to achieve a goal for each of the four baseline techniques in the longer scenario, for the case without any prior static analysis. Figure 5.9 shows the same information, but for the case where the information from a static analysis was made available.

Discussion

The baseline category of techniques have markedly different performance in each scenario. For the shorter scenario, the informed techniques using the best parameter values from a static analysis of the state space all improve performance compared to the uninformed techniques. This is not the case for the longer scenario, where only the B2b technique improves the result compared to the uninformed B2 technique.

There are several reasons for this, but the most significant is that in the shorter scenario, almost any combination of parameter values results in a plan being computed with reasonably low t_P, whereas in the longer scenario there are many combinations that exceed $15s$. In particular, the best combination of parameters found in the static analysis have very poor

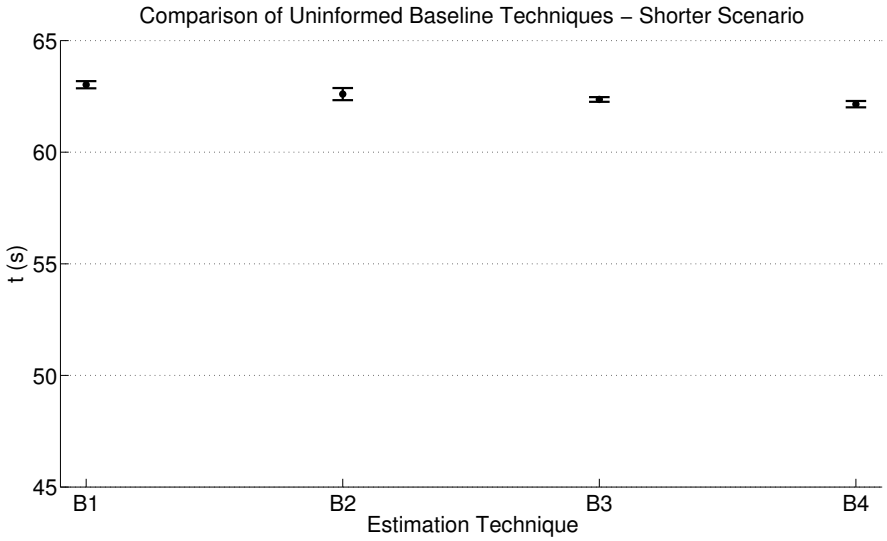


Figure 5.6: Total time to achieve a goal for the four uninformed baseline comparison techniques in the shorter planning scenario.

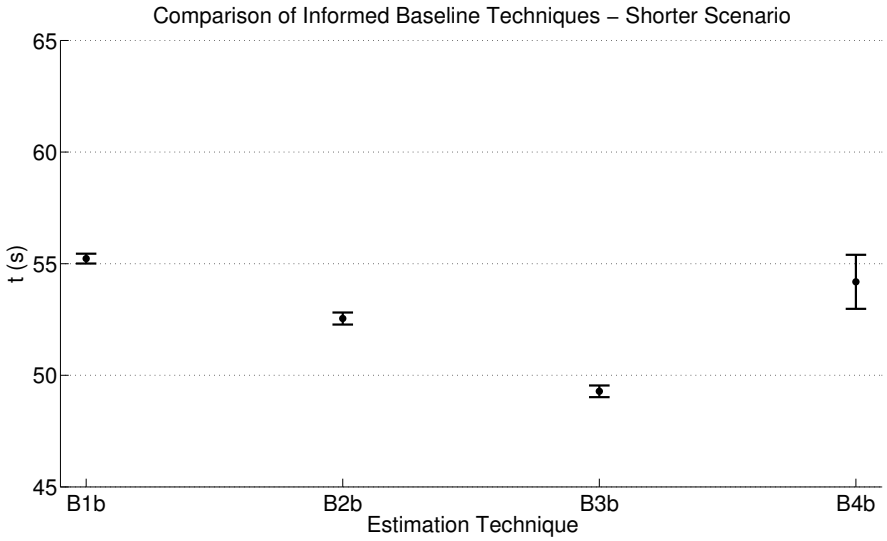


Figure 5.7: Total time to achieve a goal for the four informed baseline comparison techniques in the shorter planning scenario.

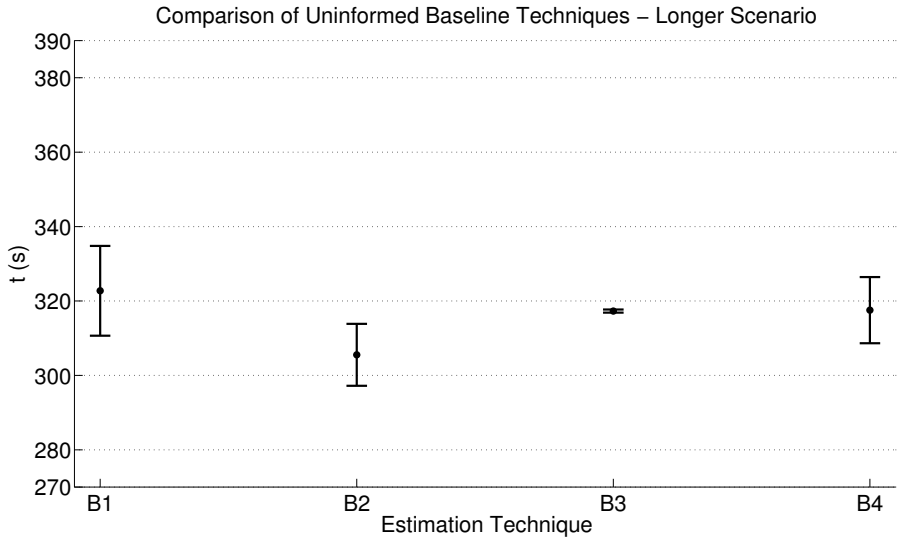


Figure 5.8: Total time to achieve a goal for the four uninformed baseline comparison techniques in the longer planning scenario.

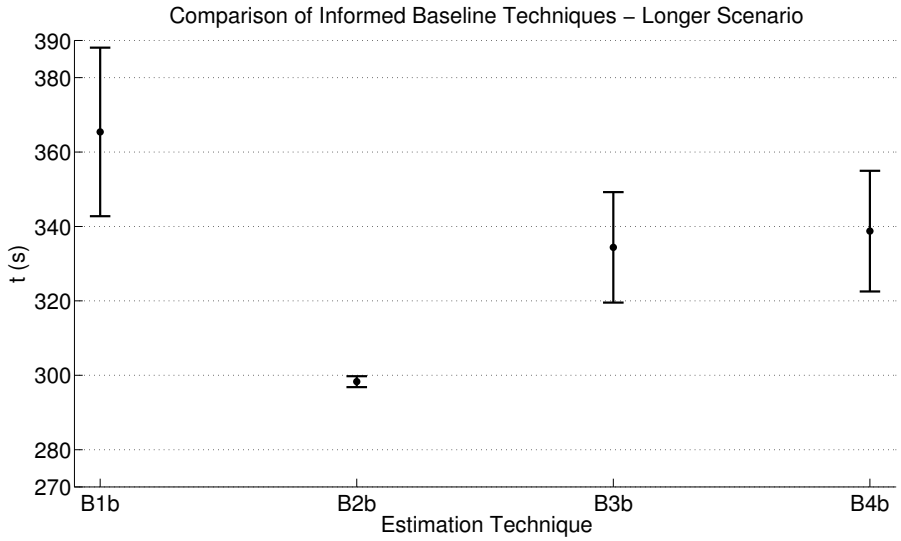


Figure 5.9: Total time to achieve a goal for the four informed baseline comparison techniques in the longer planning scenario.

performance in practice, as shown by the results of the B1b technique in the longer scenario. This reiterates the point that a fixed set of parameter values is incapable of accounting for the full range of planning system trade-offs required to maximise performance in dynamic state spaces.

5.3.2 Heuristic Results

Shorter Scenario

Figure 5.10 shows the mean total time to achieve a goal for each of the four heuristic techniques in the shorter scenario, for the case without any prior static analysis. The results are compared against the four uninformed baseline techniques. Figure 5.11 shows the same information, but for the case where the information from a static analysis was made available.

Longer Scenario

Figure 5.12 shows the mean total time to achieve a goal for each of the four heuristic techniques in the longer scenario, but only for those techniques that were successful; H1, H1b, H4, and H4b. Since both informed and uninformed techniques are shown, they are compared against all eight baseline techniques.

Discussion

Like the baseline techniques, the heuristic techniques also have distinct performance characteristics across the two scenarios. The exception to this is technique H1, and its informed counterpart H1b, both of which offer close to the same performance as the baseline techniques B1 and B1b respectively. This is unsurprising since the only change is to incorporate the use of a fast initial plan to get the agent active quickly.

Techniques H2 and H2b have poor performance compared to their respective informed and uninformed baseline comparisons. The reason for this is that they operate like an anytime algorithm but deflating the cell size rather than heuristic inflation factor, and the criteria for deflation is ineffective. This criteria is whether $t_{P_{i-1}} \leq t_{P_{max}}$, with $t_{P_{max}} = 0.1s$, and

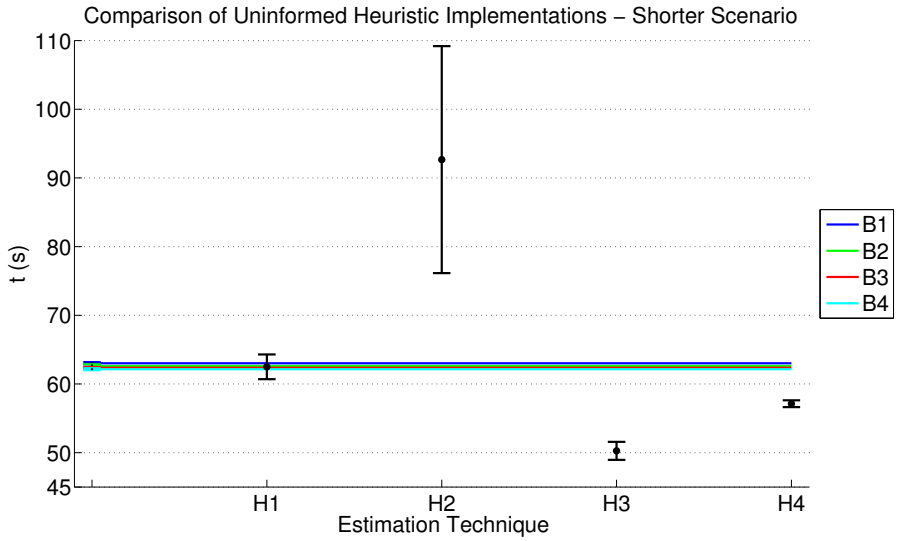


Figure 5.10: Total time to achieve a goal for the four uninformed heuristic techniques in the shorter planning scenario.

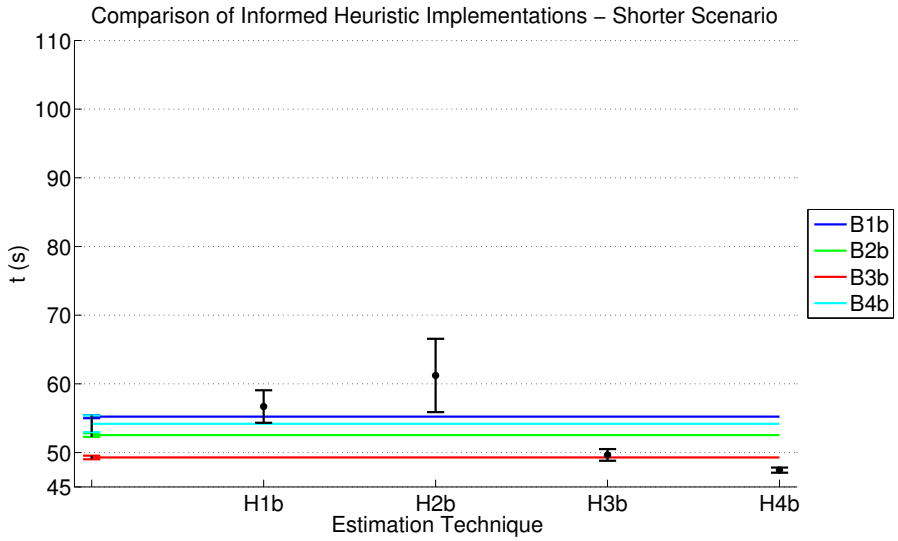


Figure 5.11: Total time to achieve a goal for the four informed heuristic techniques in the shorter planning scenario.

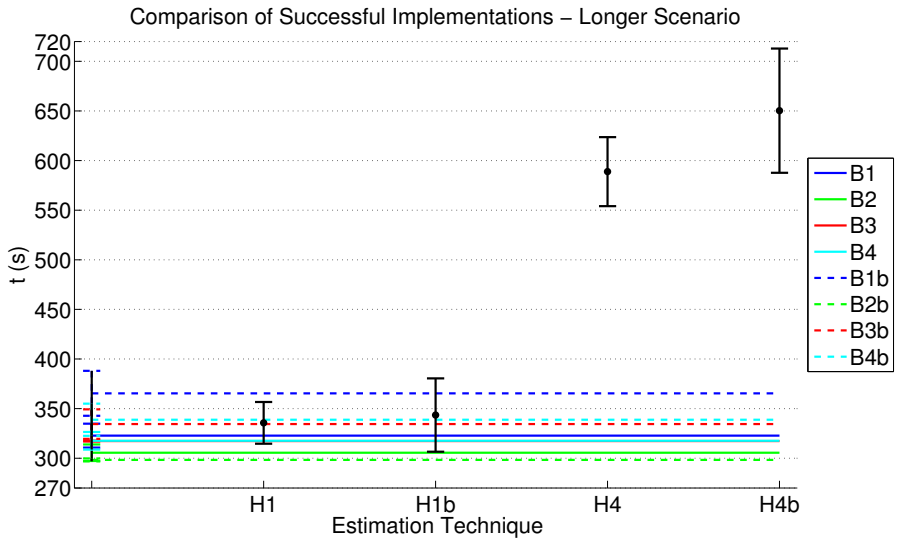


Figure 5.12: Total time to achieve a goal for the four successful heuristic techniques in the longer planning scenario.

the fact that the techniques have poor performance suggests that there are situations in which the time taken to compute a detailed plan is necessary, and that favouring t_P over t_E is not always appropriate.

In the longer scenario, neither of the techniques H2 or H2b succeeded at all, and observation of their operation showed that a cell size below $2m$ was never selected. This suggests that there are no plans (at least early in the scenario) with a cell size below $2m$ for which $t_P \leq 0.1s$. Performance may have improved if $t_{P_{max}}$ was increased from $0.1s$, however this would require tuning of the technique for different operating scenarios, which is undesirable since one of the main objectives of the TOPE process is to avoid scenario-specific parameter adjustments.

Techniques H3 and H3b also use this same criteria, but allow deflation of both the cell size and heuristic inflation factor. When a deflation causes $t_{P_{i-1}}$ to exceed $t_{P_{max}}$, then the other parameter is deflated instead. The results show this to be a particularly effective strategy in the shorter scenario, but completely ineffective in the longer. The reason for this is that the strategy is a form of gradient descent, but the surface being assessed is just $t_{P_{i-1}}$, not the appropriate metric surface of $\hat{t_P} + \hat{t_E}$. In the shorter scenario, the parameters selected by the strategy happen to be effective, but the unsuccessful result in the longer scenario shows that the heuristic is not always appropriate.

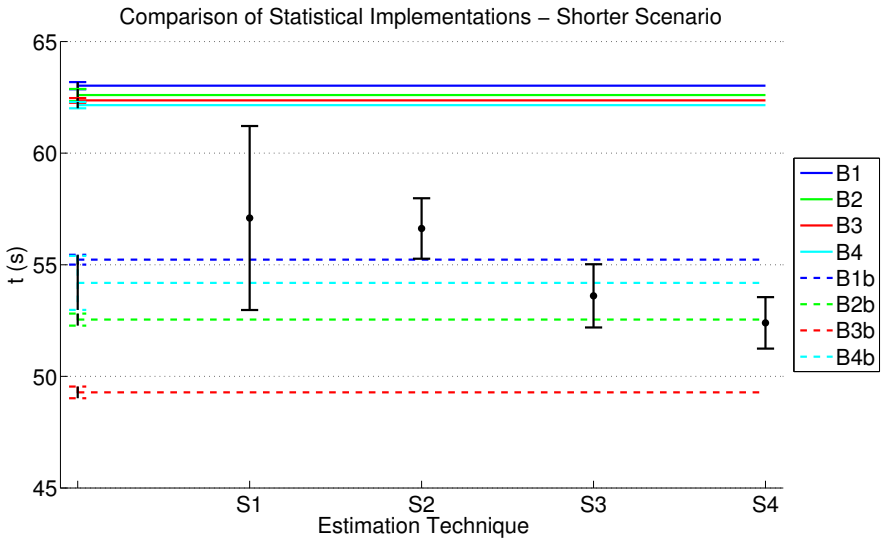


Figure 5.13: Total time to achieve a goal for the four statistical techniques in the shorter planning scenario.

Techniques H4 and H4b are similar to H3 and H3b, but the criteria used to switch the parameter being deflated is the difference in total $t = t_C + t_P + \hat{t_E}$ between the two previous iterations. This means that the surface being assessed is the appropriate metric surface, but although performance is effective in the shorter scenario, it is successful but very poor in the longer scenario. The reason for this poor performance in the longer scenario is that the metric surface can (and did) have local minima, and the gradient descent form of technique H4 results in the parameter values becoming stuck within this minima. Again, this implies that while heuristic techniques may be appropriate in some scenarios, they may have poor performance in others.

5.3.3 Statistical Results

Shorter Scenario

Figure 5.13 shows the mean total time to achieve a goal for each of the four statistical techniques in the shorter scenario, none of which use information from the static analysis. The results are compared against the four uninformed baseline techniques.

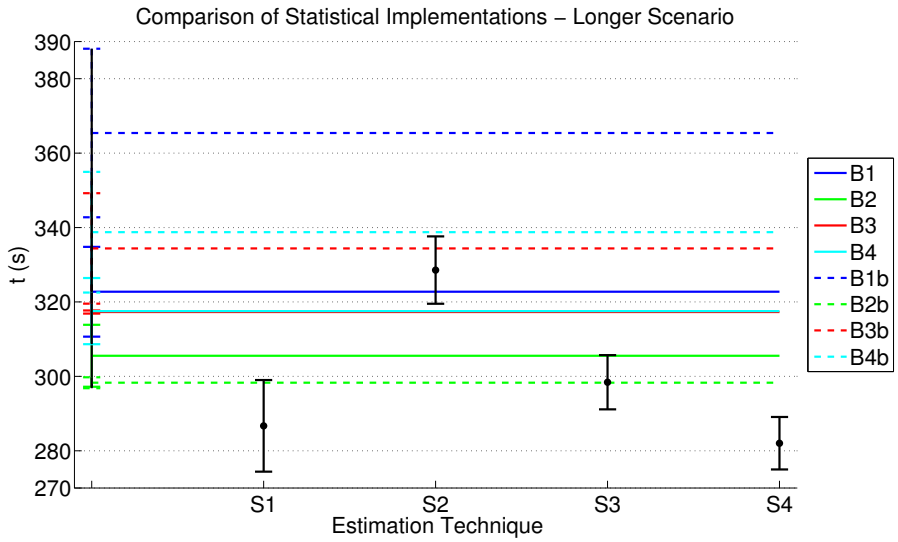


Figure 5.14: Total time to achieve a goal for the four statistical techniques in the longer planning scenario.

Longer Scenario

Figure 5.14 shows the mean total time to achieve a goal for each of the four statistical techniques in the longer scenario. The results are compared against the four uninformed baseline techniques.

Discussion

In the shorter scenario, all four statistical techniques result in a total time to achieve a goal of significantly less than any of the uninformed baseline techniques. Techniques S1 and S2 have similar performance, but the addition of a time limit of $t_{P_{max}} = 0.1s$ in S2 results in a lower variance in the results, since this mitigates the detrimental effect of selecting random parameters that require a high planning time.

For the longer scenario however, the results are completely different. Technique S1 again results in better performance than the uninformed baseline techniques, however S2 is significantly worse. The reason for this is the same as that which explains the poor performance of the heuristic techniques with planning time limits; there are some circumstances in which slow but detailed plans are more appropriate.

Technique S3 has good performance in both scenarios, outperforming all techniques except B2 in the longer scenario, where it is still superior on average but has too large a variance for this to be regarded as significant. Technique S4 however improves upon S3 again, with the total time required to achieve a goal significantly lower than all four uninformed baseline comparison techniques in both scenarios. S4 also has the lowest variance of all techniques.

Comparisons with the informed baseline techniques are also favourable, despite the informed techniques having additional information that is unavailable to the statistical techniques. All the techniques are superior to B1b in the longer scenario, and S3 and S4 are superior to it in both. In the shorter scenario, S4 has similar performance to B2b (but has a higher variance), which is explained by the trace in Figure 4.8 which showed that the optimum cell size is almost constant in the shorter scenario, meaning B2b is highly effective since it uses this cell size value. In the longer scenario, S1 and S3 are superior to all the informed baseline techniques except B2b.

Technique S4 is the clear winner of all the statistical techniques. In the longer scenario it is superior to all of the baseline techniques, both informed and uninformed. In the shorter scenario it is superior to all the uninformed baselines, as well as B1b. The key features are that it achieves this performance without any *a priori* information, and without the manipulation of any tuning parameters. Although it cannot be verified without exhaustive analysis of alternative scenarios, it is suggested that these features imply that S4 and techniques like it may be applicable to many other planning scenarios and systems.

5.3.4 Conclusions

This section discusses the results of the assessment of each type of TOPE estimation technique, but focusses on how the different categories perform and compare with each other, rather than individual techniques within categories. To aid comparison between the different categories, Figures 5.15 and 5.16 show all the successful results on single graphs, for the shorter and longer scenarios respectively.

As described in Section 5.2.2, the domain knowledge encapsulated in the heuristic techniques was drawn predominantly from the experiments performed in Chapter 4. Since these experiments used only the shorter scenario, it was to be expected that the heuristic techniques would have their best performance there too. In the shorter scenario, it is seen that the best

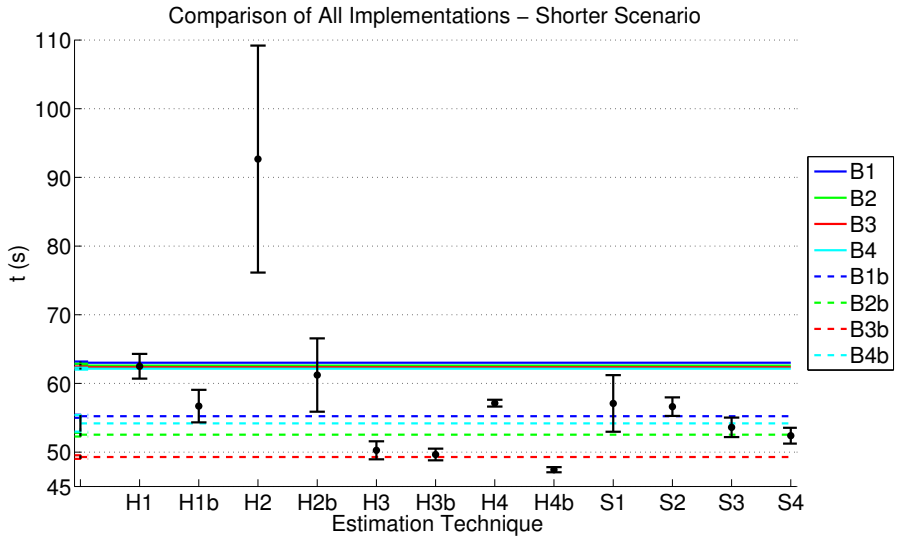


Figure 5.15: Total time to achieve a goal for all techniques in the shorter planning scenario.

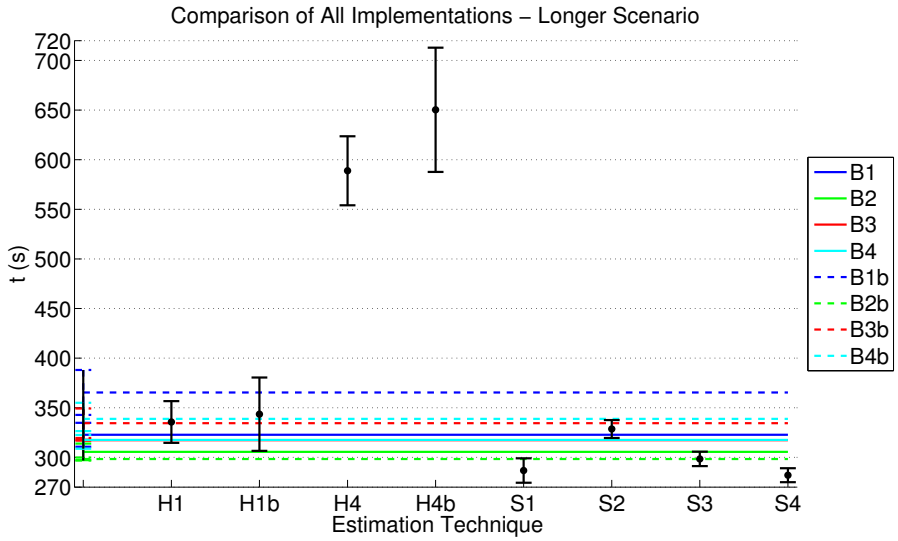


Figure 5.16: Total time to achieve a goal for all successful techniques in the longer planning scenario.

performing heuristic techniques (H3, H3b, and H4b) all have better performance than the statistical techniques S1-4. This suggests that when the heuristics are most effective, then performance is likely to be superior to techniques that do not make assumptions about the structure of the scenario. However, when these same heuristic techniques are used in the longer scenario, the performance is far worse than any other techniques. By comparison, the statistical techniques have consistent performance, with S4 seen to be the best of the category in both scenarios.

Differences in the performance of the baseline techniques are also apparent between the two scenarios. In the shorter scenario, the informed techniques are all better than the uninformed, but this is not the case in the longer scenario. The reasons for this were discussed above, but are all related to the fact that as the scope of the scenario is increased, the chances also increase of there being situations in which the best performing parameters from analysis of a static state space are unsuitable.

The main distinction between the categories is that the statistical methods do not require tuning parameters, or any *a priori* domain knowledge, whereas all the other techniques do. Both the informed and uninformed baseline techniques require suitable choices of fixed parameter values, and even the uninformed heuristic techniques require some knowledge in order to determine the appropriate heuristics. Analysis of the techniques used in the fielded planning systems described in Chapter 2 suggests that it is almost always possible to derive appropriate heuristics, but that it is seldom easy to perform the static analysis of Chapter 3 to determine the best performing choice of parameter values in a static state space.

The statistical techniques S1-3 were seen to have good performance in both scenarios without using any domain knowledge. When domain knowledge was available, and appropriate heuristics could be determined, technique S4 showed that this performance could be further improved, again in both scenarios.

The conclusions that can be drawn from this are that where it is possible to find domain knowledge that can be encapsulated in a suitable heuristic, the heuristic techniques may have good performance. However, slight changes to the scenario may mean the heuristics become inappropriate, and so this level of performance cannot be guaranteed by heuristic techniques. Statistical techniques can be used with or without domain knowledge, and have better performance when they can be augmented with the strategy of a heuristic technique. This means they can be used across a wide variety of scenarios and planning systems,

including in completely unknown scenarios where no other technique can have a justifiable rationale for the selection of particular fixed parameters.

5.4 Monte-Carlo Analysis of TOPE Estimators

The experiments in Section 5.3 provided a detailed analysis of the performance of various TOPE estimators in the two example scenarios. This section further analyses the performance of these techniques in a far wider range of scenarios, aiming to show that the results obtained in the example scenarios are indicative of likely performance in others.

Section 5.4.1 describes the Monte-Carlo procedure used for this analysis, the construction of randomised fractal terrain for use as cost data, and the measures used to assess performance. Section 5.4.2 presents the results of the Monte-Carlo procedure applied over real world cost data (as visualised in Figure 2.1), and Section 5.4.3 presents the results of the procedure applied over randomised fractal terrain with three different values of terrain roughness. The full result tables and performance matrices (described below) are provided in Appendices C and D, respectively. Finally, Section 5.4.4 discusses the results of all the Monte-Carlo experiments, and the conclusions that can be made regarding the relative performance of TOPE techniques compared to existing state-of-the-art techniques from the literature.

5.4.1 Experimental Procedure

The experiments in this section take the form of a Monte-Carlo analysis, in that the input scenarios or objectives are randomised and performance is assessed over a large set of experimental runs. The TOPE estimators considered are those described in Section 5.2.

Each experimental run begins with the agent in a randomised starting pose, with a randomised final goal, subject to the constraints that both points are non-obstacles for all the cell sizes in the set of cost maps, and the distance between the start and goal is greater than $20m$ and less than $500m$. It should be noted that given these constraints, it is possible for a particular baseline technique to fail entirely if the start and goal points are blocked by obstacle cells in the baseline technique's fixed cell size cost map. This point itself is indicative of the benefits of TOPE techniques that allow adjustment of system parameters, over techniques that rely on a fixed parameter set.

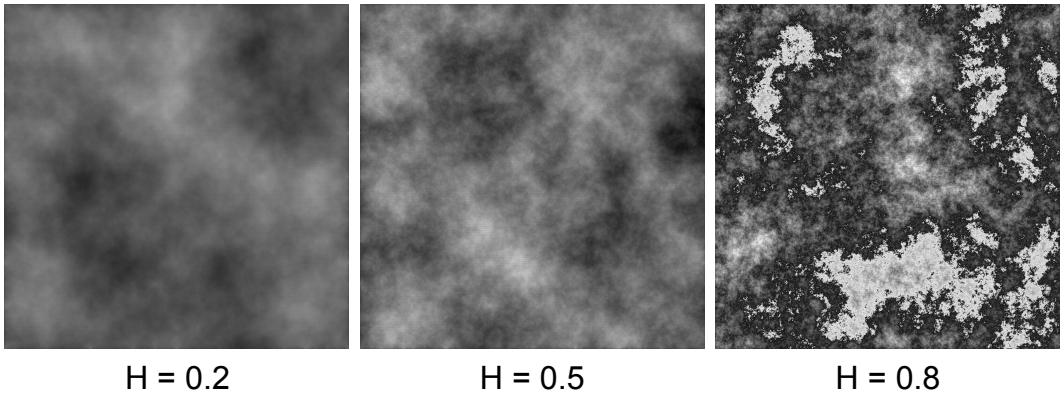


Figure 5.17: Example randomised fractal cost maps for various values of the roughness parameter, H. From left to right, the values of H are 0.2, 0.5, and 0.8, and the rate of change of cost values along any spatial direction is seen to increase with the roughness parameter value.

In the first set of experiments, the cost maps are again generated from the data visualised in Figure 2.1. In the second set of experiments, the cost maps are generated from randomised simulated data using a fractal terrain generator known as the diamond-square algorithm (Fournier et al. 1982). One of the key features of this algorithm is a parameter, H, which affects the roughness of the output by allowing more or less random variation in the value of each pixel (Miller 1986, Voss 1987). In the second set of experiments, values of $H = 0.2$, $H = 0.5$, and $H = 0.8$ are used to provide low, medium, and high levels of roughness. Figure 5.17 demonstrates example cost maps for each of these values of H, showing that the rate of change of cost values in any spatial direction increases with the value of H. As for the real world data visualised in Figure 2.1, the randomised cost maps are generated at a far higher resolution than any of the cell sizes in the TOPE parameter space. They are then converted using the pessimistic filter described in Section 2.2.1 and shown in Figure 2.3.

For each experimental run, a brute-force static analysis is performed for the 1D parameter space consisting only of the cell size, c, and also for the 2D parameter space consisting of the cell size and heuristic weighting factor, ε. The results of this static analysis are then automatically applied to the informed techniques. Each baseline and TOPE estimator technique is then run for 5 repetitions, and the total time taken is recorded for each. For

the second set of experiments, a newly randomised fractal cost map is generated for each run, again with 5 repetitions of each technique performed for each map.

To compare across the different experimental runs, the difference in total time taken between each TOPE estimator and each baseline technique are assessed. A technique is considered to 'outperform' another if its total time taken is lower, or it succeeds where the other fails. Due to the fact that the techniques may fail to complete an experimental run, averaging the performance improvement of one technique compared to another is not possible, and a probabilistic distribution of the likely performance improvement of each technique cannot be determined.

A frequentist analysis is performed instead, and records the percentage of runs in which each TOPE estimator outperformed each baseline technique. The last row in the tables of these results (labelled 'Min') records the percentage of runs in which each TOPE estimator outperformed the best performing baseline technique in each particular run. In practice, one could not know this information in advance, and thus this row effectively shows the worst-case performance benefit of each TOPE estimator, compared against a hypothetical system that just happens to use the best possible non-TOPE technique in each run. The analysis is also performed for different margins of performance improvement, i.e. the percentage of runs in which each TOPE estimator outperformed each baseline technique by a given percentage of the baseline technique's total time. Tables are shown for margins of 5%, 10%, and 25%.

In addition, the comparison is also performed the other way around, recording the percentage of runs in which each baseline technique outperformed each TOPE estimator. The last column in the tables of these results (again labelled 'Min') records the percentage of runs in which each baseline technique outperformed the best performing TOPE estimator in each particular run. Again, this analysis is also performed for different margins of performance improvement, i.e. the percentage of runs in which each baseline technique outperformed each TOPE estimator by a given percentage of the baseline technique's total time. As before, the tables are shown for margins of 5%, 10%, and 25%.

For each of the four sources of cost data; real world data, and randomised fractal terrain with roughness values of $H = 0.2$, $H = 0.5$, and $H = 0.8$, there are eight tables in total. These are shown in Appendix C in Sections C.1 through C.4, respectively.

5.4.2 Monte-Carlo Analysis over Field Trial Cost Map Data

This section presents the results of the Monte-Carlo analysis for the real-world cost map data visualised in Figure 2.1. The eight tables of results described in the previous section are shown in Appendix C in Section C.1.

Table C.1 shows the percentage of runs in which each comparison method outperformed each baseline technique. Table C.2 shows the reverse; the percentage of runs in which each baseline technique outperformed each comparison method. It is clear that overall, the percentages shown in Table C.1 are generally higher than those shown in Table C.2. This indicates that the TOPE estimators are more often superior to the baseline methods than the opposite. The tables also show that the best performing TOPE estimator is still S4 (as noted in Section 5.3.4) and that the best performing baseline technique is B3. From Table C.1, S4 has equal or better performance to B3, 59.4% of the time, however from Table C.2, B3 has equal or better performance than S4, only 38.8% of the time. Looking at the column and row labelled 'Min' in each table respectively, it can be seen that S4 outperformed the best possible baseline technique 45.1% of the time, whereas B3 outperformed the best possible TOPE estimator just 8.4% of the time, for scenarios within this cost map data set. These statistics indicate that more often S4 is a better choice of technique compared to B3, than vice-versa.

Tables C.3, C.4, and C.5 show the percentage of runs in which each TOPE estimator outperformed each baseline technique by margins of 5%, 10%, and 25% of the baseline technique's total time (or the baseline technique failed to complete the run and the TOPE estimator succeeded), respectively. Likewise, Tables C.6, C.7, and C.8 show the percentage of runs in which each baseline technique outperformed each TOPE estimator by these same margins (or the TOPE estimator failed to complete the run and the baseline technique succeeded).

Looking at just the cells for the best performing baseline technique, B3, compared to the best performing TOPE estimator, S4, for example, the tables show that S4 is able to outperform B3 by 5% or more, 40.3% of the time, by 10% or more, 28.4% of the time, and by 25% or more, 16.7% of the time. Conversely, B3 is only able to outperform S4 by 5% or more, 17.0% of the time, by 10% or more, 7.2% of the time, and by 25% or more, merely 3.0% of the time. These statistics indicate that the average performance of S4 is significantly better

than the average performance of B3, for randomised scenarios over the real-world cost map data. Not only is S4 more often a better choice than B3, the performance improvement when it is better is far more significant than the performance deterioration when it is worse.

This level of detailed analysis can be drawn from the tables for any combination of a TOPE technique and baseline technique, but the main outcome is simply to confirm whether the TOPE technique outperforms the baseline more often that the opposite, and whether this holds at various margins of performance improvement. The following figures present performance matrices for each combination of TOPE technique versus baseline technique that summarise this outcome for each combination of techniques. Cells in the performance matrices are green if the TOPE technique outperformed the baseline more often than the opposite, yellow if both techniques outperformed each other equally as often, and red if the baseline technique outperformed the TOPE technique more often than the opposite.

Four such matrices were generated, for margins of 0%, 5%, 10%, and 25%. All of these matrices are provided in Appendix D in Figures D.1 through D.4, and the zero margin case is shown in Figure 5.18 below. In the performance matrices, a column which is predominantly green in all figures indicates a TOPE technique which outperforms all alternative techniques for every performance margin. For example, the figures indicate that over the real world data, S4 is such a technique, and this can be confirmed by reference to Tables C.1 to C.8.

5.4.3 Monte-Carlo Analysis over Simulated Cost Map Data

This section shows the results of the same Monte-Carlo experiment, but performed over the randomised fractal terrain, with roughness parameter values of $H = 0.2$, $H = 0.5$, and $H = 0.8$. The tabulated results for each of these roughness parameter values are provided in Sections C.2, C.3, and C.4, respectively. The performance matrices for each margin are provided in Appendix D, with only the zero margin cases for each roughness parameter value shown in this section, in Figures 5.19, 5.20, and 5.21, respectively.

As for the real world data, the tables again show that, on average, B3 is the best performing baseline technique, and that S3 and S4 are the best performing TOPE techniques, for all three roughness parameter values. The performance matrices show that S3 and S4 outperform nearly all the baseline techniques more often than they are outperformed by them, and that this holds for all the performance margins assessed. Overall, these results

indicate that S3 and S4 offer superior performance to all the baseline comparison techniques, across a wide range of scenarios and mission lengths.

5.4.4 Discussion

The Monte-Carlo analysis presented in this section compared the performance of 12 possible TOPE techniques against 8 baseline comparison techniques from the literature, over a wide variety of mission scenarios and cost data. While it is difficult to determine a distribution from which expected performance of each technique could be drawn, it is trivial to compare the performance of any two techniques. For almost all of the possible pairs of techniques, in all four categories of cost data, the TOPE techniques were found to offer superior performance more often than the comparison technique. When comparing the best performing of the 12 TOPE techniques against the best performing of the comparisons, this result still holds. From these results, it is concluded that a suitable TOPE technique is more likely to outperform than be outperformed by any comparison technique.

Furthermore, the TOPE techniques had better margins of performance improvement more often that the comparison techniques. In other words, the percentage of times that a particular TOPE technique outperformed a given comparison technique by a given margin, was significantly greater than the number of times that the comparison technique outperformed the TOPE technique by this same margin. From these results, it is concluded that the average performance of a suitable TOPE technique across many scenarios is likely be superior to the average performance of any comparison technique, since the times that it is beneficial will typically outweigh the times that it is detrimental.

The TOPE techniques can also be said to be more robust than the comparison techniques, in that they generally succeed in a wider range of scenarios. For the experiments using real world data, in 9.25% of the runs all 8 of the comparison techniques failed, while at least one of the 12 TOPE techniques succeeded. By comparison, not once did all 12 TOPE techniques fail while at least one comparison succeeded. For randomised fractal terrain with roughness parameters of 0.2, 0.5, and 0.8, the frequencies of all comparisons failing with one more TOPE techniques succeeding versus all TOPE techniques failing with one or more comparisons succeeding were 9.92% versus 1.65%, 17.62% versus 1.64%, and 15.38% versus 4.07%, respectively.

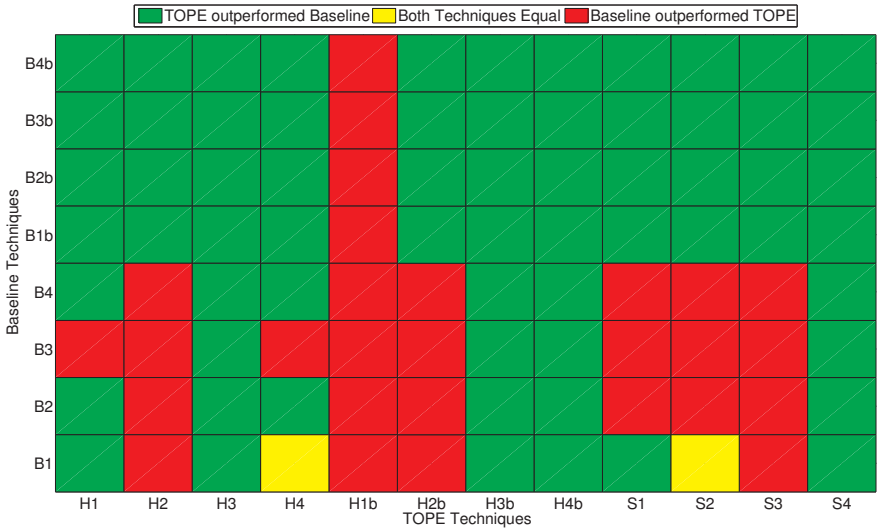


Figure 5.18: Performance matrix comparing TOPE techniques against baseline techniques over real world data, with no performance margin.

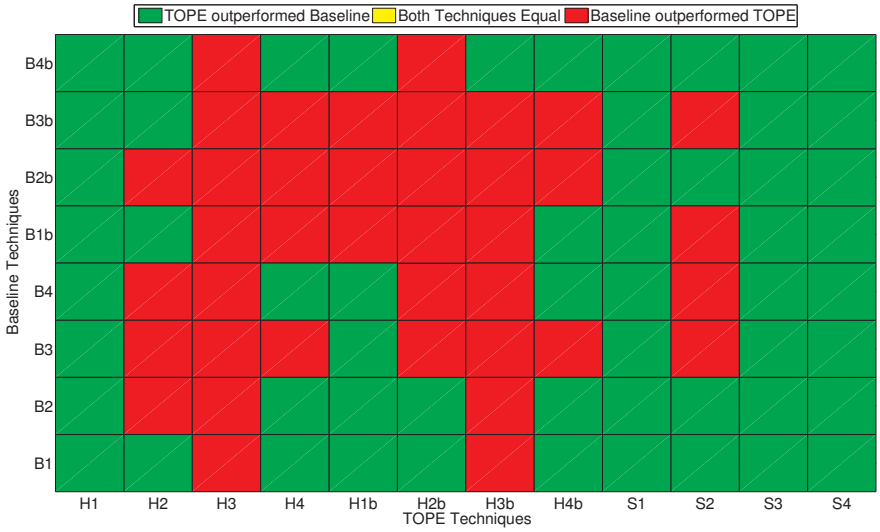


Figure 5.19: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.2$, and with no performance margin.

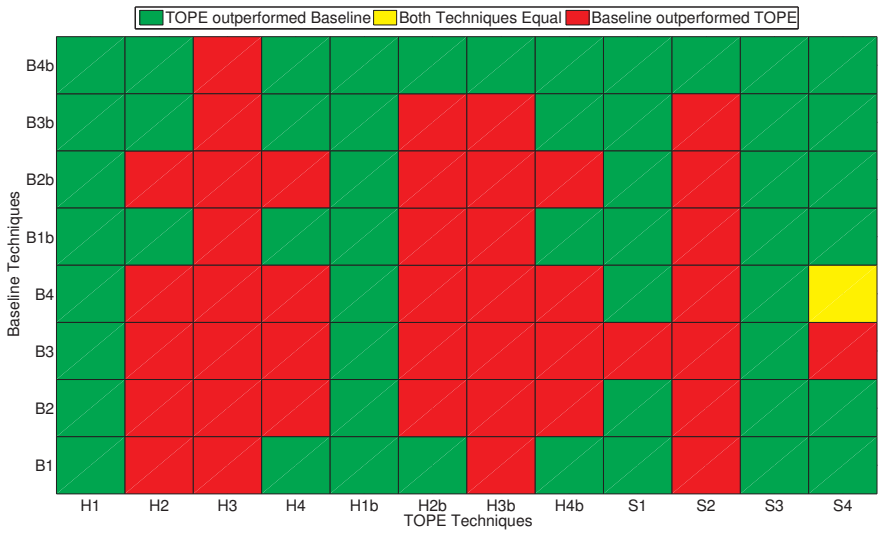


Figure 5.20: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.5$, and with no performance margin.

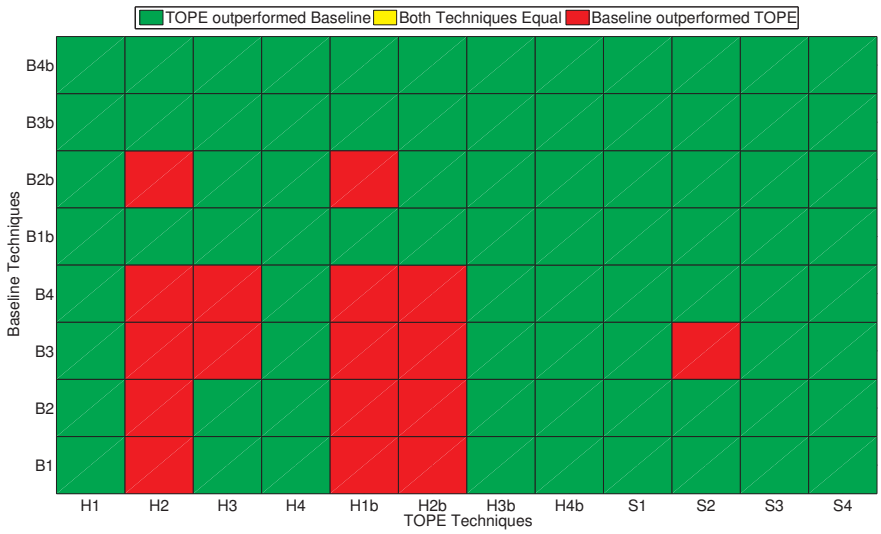


Figure 5.21: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.8$, and with no performance margin.

These conclusions show that, on average, TOPE techniques are superior to existing techniques that use fixed parameter values in three main areas: they are more often beneficial than detrimental; they are beneficial by a greater amount than they are detrimental; and they are more robust across a wider range of scenarios. Overall, this section has shown that TOPE techniques are a provably effective technique for reducing the total time taken to achieve a goal, within planning systems for autonomous agents.

5.5 Conclusion

This chapter presented two categories of TOPE estimation techniques that were shown to be superior to existing techniques using fixed parameter values, under certain conditions. These techniques – heuristic and statistical TOPE estimators – were defined, and several examples of each were developed. The experiments presented within this chapter then compared these techniques to existing fixed parameter techniques in the literature, aiming to verify the original thesis statement from Chapter 1:

That the TOADM process is able to meet or improve upon the performance of any method that uses only fixed system parameters, given the same sources of state information.

The results presented in Section 5.3 verify this hypothesis for the TOPE problem, by validating it in the example problem scenarios. Informed heuristic techniques were shown to be capable of outperforming informed fixed parameter techniques, and even anytime techniques starting from the same state information. Uninformed statistical techniques were shown to be capable of outperforming all uninformed fixed parameter techniques, anytime techniques, and in some cases even informed fixed parameter techniques that start with additional state information.

It is impossible to verify this thesis statement for all problem scenarios and systems. However, the results presented in Section 5.3 verify it for at least some scenarios within the TOPE problem, and suggest that it may be true in others. The results of the Monte-Carlo analysis in Section 5.4 further show that the average performance of TOPE estimators is superior to all the baseline comparison techniques that use only fixed parameter values, across a wide range of mission lengths over real world data, and wide range of simulated scenarios using randomised fractal terrain. The superiority of the TOPE techniques was

demonstrated in three main areas: they were shown to be more often beneficial than detrimental; beneficial by a greater amount than they are detrimental; and more robust across a wider range of scenarios.

With reference to planning systems for autonomous agents, these results indicate that the use of the TOPE framework in conjunction with statistical estimators such as S3 or S4 can offer significant performance improvements to existing planning systems such as the PHR. It is suggested that incorporating the TOPE estimation techniques described in this chapter into the existing PHR planning system would result in a completely autonomous planning system, free from any tuning parameters, and with performance significantly superior to comparable state-of-the-art systems currently in operation.

Furthermore, it is important to reiterate that the TOPE process is applicable to any planning system with dynamically adjustable system parameters. It does not require specific algorithms or parameter sets, but offers a framework within which existing components can be manipulated to give the best overall performance, using a time-focussed metric. If further research finds a novel and better performing algorithm or data structure, it is likely that the TOPE process will be able to manipulate the tuning parameters in this technique to maximum benefit. If advances in processor technology allow more computationally intensive processes to produce closer to globally optimal performance, again the TOPE process can dynamically adjust tuning parameters to suit an improved processing power.

The strength of the TOPE process is not in the performance improvements shown in the examples within this chapter, but in the potential that the framework has for all systems in which performance is principally measured by the time taken to achieve a goal.

Chapter 6

Conclusions and Future Work

This thesis has presented a novel contribution to the field of decision making problems, in particular for those in which performance is principally measured by the time required to complete objectives. For problems where the amount of available computational power is insufficient to solve them completely, trade-offs must be made to render each problem tractable. The TOADM framework then provides a means of making these trade-offs in a manner that minimises the total time required to complete the objectives.

Since the TOPE equation encompasses the solution of the intractable planning problem, it is clear that the TOPE problem itself is intractable, and cannot be solved analytically. Thus, the solution is presented in this thesis by way of a procedure instead. Furthermore, this procedure cannot be validated analytically since it is dependent on the nature of the problem space in which it is used. Thus, the claim in this thesis is only that the procedure is *capable* of meeting or improving upon the performance of alternative methods, not that it will *always* improve upon them. This claim can be verified by a single example, as was done in Chapter 5. In addition, the procedures from Chapter 4 provide methods to determine whether the TOPE process will be suitable for any particular problem space.

The contributions of this thesis are summarised in Section 6.1. Section 6.2 then demonstrates how each of the TOPE-specific contributions can be generalised to the TOADM problem. Section 6.3 discusses some considerations pertaining to TOADM implementations, and finally Section 6.4 proposes directions for future work.

6.1 Summary of Contributions

6.1.1 Time-Focussed Metrics

Chapter 2 presented a historical overview of research in the planning domain, and showed that most of this work involved improvements to the data structures and algorithms that reduced either the planning or execution time required, but seldom both together. The concept of time-focussed metrics was described, whereby each parameter in the performance metric used by the system is a measure of time. The sum of each of these measures is then itself an uninflated measure of time, making the overall function a time-focussed metric. If the final value of this metric represents the expected time required to complete the objectives of the system, then it can be used as feedback to an ADM in order to create a TOADM system.

6.1.2 Dynamic Tuning Parameters for Planning Systems

Chapter 3 described several categories of tuning parameters found in existing planning systems, including state space discretisations, sampling density, heuristic weightings, path diversity, and others. Examples of these categories were analysed, and it was shown that the total planning and execution time varied significantly across the span of parameter values assessed. It was further shown that combinations of parameters can be analysed in the same manner, if a time-focussed metric is employed to assess performance of the planning system. Dynamic adjustment of these parameters while a planning system is active can be used to create a TOPE system.

6.1.3 Admissible Anytime Performance Bounds

Section 3.4 described a technique that applied a time-focussed metric to an active anytime planning process. The technique was able to determine an admissible bound on the expected optimal performance of the process, while the process was active. This bound could be used to determine when an anytime process was incapable of improving performance, but this was shown to be unsuited to dynamic state spaces. The admissible bound itself however has use in applications such as operator displays that indicate the expected time remaining

to achieve a goal, or estimation processes in which knowledge of the maximum benefit to be obtained may be relevant.

6.1.4 Derivation of the TOPE Equation

Section 4.1 presented a time-series depiction of a replanning process in a dynamic state space, and showed the accumulated time taken and expected time remaining at each iteration. This analysis was used to derive the total time required by the replanning process. Minimisation of this total time was shown to be the objective of the TOPE problem, and the final TOPE equation was derived as:

$$y^* = \arg\min_{y \in Y} \left\{ \hat{t_C} + \hat{t_P} + \hat{t_E} \right\} \tag{6.1}$$

The equation makes explicit an easily understood intuition – the most beneficial values of system parameters are those that are expected to minimise the remaining time required to achieve a goal.

6.1.5 A General TOPE Procedure

Section 4.3 presented a general procedure known as the TOPE process, that can be applied to an existing replanning system that is structured in a similar manner to the PHR system described in Allen et al. (2009). The simplest means of doing so is to compute the result of the TOPE equation and apply this parameter set to the system just prior to each planning step. It was verified that if this procedure can be performed sufficiently quickly and with perfect accuracy, then it is capable of reducing the time required to achieve a goal, compared to any other system operating in the same state space, and using only fixed values of the system parameters.

6.1.6 A General Procedure for TOPE Sensitivity Analysis

Section 4.4 presented a second procedure to assess the sensitivity of the TOPE process to inaccuracy in the calculation of the estimated best parameters, or to variations in the time required for this estimation. It was shown that there exists an operating region in which increased estimation time can be traded for increased estimation accuracy, and vice versa.

Similarly, there exists an operating region in which the acceptable variance in estimation time can be increased if the average estimation time is decreased, and vice versa. If this sensitivity analysis is performed with reference to an existing comparison technique, then the bounds of these operating regions can be found explicitly. These bounds specify the required performance characteristics of the TOPE process such that it results in improved performance compared to the comparison technique.

6.1.7 TOPE Estimation Techniques

Chapter 5 showed that practical implementations of the estimation of system parameters in the TOPE process are found in two distinct categories; heuristic and statistical techniques. The demarkation was made on the basis of whether the techniques assessed a few or no prior results in the case of heuristic techniques, or stored all prior results in the case of statistical techniques. Analysis of several examples of each technique showed that although heuristic techniques can have superior performance in particular scenarios for which they have sufficient domain knowledge and appropriate heuristics, the same techniques often have poor performance in alternative scenarios. Statistical techniques were shown to be more robust, having effective performance in multiple scenarios. The main contribution was to show that statistical techniques can be used to solve the TOPE problem without the need for any domain knowledge, and without any *a priori* parameter tuning. Further experimentation by way of a Monte-Carlo analysis then showed that the average performance of these statistical techniques was superior to the performance of comparable methods.

6.2 Generalisation to the TOADM Problem

This section suggests that the TOPE-specific contributions presented above are not limited to the TOPE problem, but can be generalised to the broader class of TOADM problems as well. To support this assertion, the similarities between replanning and ADM problems are explored.

In a typical replanning problem for which the TOPE process is applicable, the basic procedure is to compute a plan and then execute it while computing the next plan. In an ADM problem, the procedure is to make a decision and then execute it while making the next

decision. The parallels are obvious and suggest that the TOADM equation can be defined as:

$$y^* = \arg\min_{y \in Y} \left\{ \hat{t_C} + \hat{t_D} + \hat{t_E} \right\} \tag{6.2}$$

where t_D is the time required to make a decision. The time components are thus the expected times required to compute the solution to Equation 6.2, make the decision given the argument set y^*, and execute the computed decision, respectively.

It is interesting to note that if $\hat{t_C} + \hat{t_D} = 0$, which is true only if infinite computational power is available, then the parameters are selected to minimise $\hat{t_E}$. This is equivalent to the assumptions made by problems such as time-optimal planning, in which the time required to plan is deemed unimportant, and the TOADM equation leads to the selection of exactly the same parameters in order to produce time-optimal execution. If $\hat{t_E} = 0$ then the response to decisions is instantaneous, and the parameters are selected to minimise $\hat{t_C} + \hat{t_D}$. This is appropriate for problems such as scheduling and search, which can be considered as decision making problems where the decisions themselves are the required results, and their 'execution' is outside of the process. These observations indicate that the general class of TOADM problems is broad, and encompasses many existing problem domains by removing their limiting assumptions.

Since replanning and ADM procedures have obvious parallels, it follows that the TOPE process can also be generalised to the TOADM process. The procedure requires the computation of Equation 6.2 and the application of the resulting y^* to the system prior to every decision. Similarly, sensitivity analysis can be performed by perturbing the calculated y^* prior to applying it to the system. If the state space of the decision making process can be stored in such a way that it can be 'replayed' by a simulation, then the brute-force approach to parameter analysis can also be applied to the TOADM process.

The TOPE estimation techniques can also be applied to the TOADM process, since they are derived from observations and techniques that are not planning specific. Individual heuristics require domain knowledge appropriate to particular problems, but the generally applicable heuristic of starting with the parameters that minimise $\hat{t_C} + \hat{t_D}$ is likely to be beneficial for many problems. Most importantly, the statistical techniques shown to be valuable for the TOPE problem are derived from reinforcement learning and related methods. These methods have been used in a wide variety of problem domains, and can be

expected to suit the TOADM problem well, since they do not require any domain knowledge or *a priori* parameter tuning.

6.3 Implementation Considerations

The experimental results presented in this thesis were primarily drawn from the CORD UGV system simulator. This is mainly because of the need to perform repeatable experiments, which are difficult in practice since a skid-steered vehicle will alter the terrain properties as it drives, which will alter the way it drives in subsequent runs. In addition, the CORD UGVs were decommissioned in early 2010. Despite the absence of experimentation on a physical robot, the CORD UGV simulator shares all its interfaces and most of its software with the real system, and has had several years of development and comparison with the physical UGVs. The result is that implementation of the TOPE estimators on the real CORD UGVs would be near trivial, requiring only that multiple cost maps be stored at each resolution required, and retained in synchronisation with one another[1]. One of the main benefits of the TOPE process is that it naturally accounts for the level of computational power, and thus there would be no system parameters to tune as a result of the simulator and UGVs running upon different computational hardware.

In general however, practical implementations of TOADM systems may be non-trivial, depending upon the effect of the parameters that the process is permitted to adjust. For example, storing sensory data from a robot in data stores with multiple levels of discretisation is far more complex than simply changing the value of a heuristic inflation factor just prior to planning. These considerations were briefly discussed in Section 4.2.1, where they were described as the topic of *parameter utility*.

Another consideration is the choice of whether to use statistical or heuristic TOADM estimator techniques. As was shown in Chapter 5, statistical techniques are more general and may be reusable across multiple TOADM systems, however heuristic techniques can have good performance if the heuristics are appropriate, and can be considerably simpler to design. A good rule of thumb is that if the problem lends itself to trying several approaches before settling on the best, heuristic techniques can be incrementally improved until they

[1]This was far easier to implement in the simulation environment since the data is pre-loaded into grids of each resolution from a cost map image, and the regions that would be illuminated by the vehicle's sensors are simply copied from these grids into the active data store.

are satisfactory. If the problem has many unknown aspects or only one attempt is available, then heuristics are likely to be difficult to determine, and statistical techniques will have the additional benefit that they record the effect of each parameter choice, which may be of later use.

6.4 Directions for Future Work

The research described in this thesis has provided a general framework for incorporating the effect of computational resource constraints into ADM processes, and opens up a variety of further research directions.

6.4.1 Online Adjustment of the Parameter Space

It was discussed in Section 4.2 that TOPE (and hence TOADM) parameter spaces could include parameters that affect the size or distribution of the parameter space itself. This is a result of the assumption that t_C is likely to be affected by the size or distribution. Future work is needed to assess whether this assumption is true, and if so, what strategies are appropriate to manipulate the space. All the techniques described in Chapter 5 try to limit t_C by applying limits to the amount of sampling of the metric surface that can be performed in each iteration, or else just assume t_C is sufficiently low.

A likely direction of this work, especially for the TOPE problem, is to reassess the prior plan or decision in each iteration, in light of new or changed information in the state space. It is not possible to simply reuse a prior plan or decision, but if it can be reassessed, confirmed to be suitable, and have a new $\hat{t_E}$ calculated, then it can be compared with alternative choices. Similar logic to that presented in Section 3.4 (which assessed when anytime planning was unable to further improve performance) may be able to determine when parameter estimation is unlikely to result in better performance than simply continuing with the previous plan or decision.

6.4.2 Parallelisation of the TOADM Process

In much the same way that the HR system was parallelised in the PHR system, the estimation of parameters by the TOADM process could be run in parallel with the decision

making and execution steps. This would allow multiple decisions to be made with the same set of system parameters whilst the estimation process was determining a new set. This would only be of use in a system where t_C could be expected to be greater than t_D, such as might occur when employing a high dimensional parameter space. In addition, given a parallelised framework for TOADM estimation, it would be possible to employ multiple estimation techniques simultaneously. This could mitigate some of the problems of an unbounded estimation time, at the expense of requiring additional computational resources.

6.4.3 Assessing the Effect of Computational Power

Since the TOADM process accounts for the time required to calculate Equation 6.2 and to make each decision, it implicitly accounts for the available computational power of the system. This means that the TOADM process can be used to assess the effect of changing the computational resources available. One technique to assess this in simulation is to multiply t_C and t_D by a scalar value in the range $[0, \infty)$, and observe the effect on the total time required to complete the system's objectives. More interestingly, this technique would allow the simulation of computational resources changing availability throughout the scenario, which may be of interest in problems such as distributed computing (Andrews 2000, Ghosh 2007), swarm robotics (Fitch & Butler 2008), or decentralised systems which combine computational resources to make decisions (Tsitsiklis 1984, Matthews 2008).

6.4.4 Comparison of TOADM with Related Fields

It was suggested in Chapter 4 that the TOPE equation (and hence the TOADM equation) appears similar to a policy used by the Bellman equation (Bellman 1957*a*, chap.III):

$$V(x) = \max_{a \in \Gamma(x)} [F(x, a) + \beta V(T(x, a))] \tag{6.3}$$

This equation says that the optimum reward, V, available in a state, x, depends on a function over all the actions, a, that are possible given the constraints of the current state, $\Gamma(x)$. This function is the sum of the reward from taking action a in state x, $F(x, a)$, and the reward, discounted by factor of β, available in the state into which action a transitions from x, which is given by $T(x, a)$.

By calculating the value function, V, a related function, $a(x)$, is also found. This function is known as a policy, since it describes the optimal action as a function of the state. The TOADM equation can be considered as a policy which determines the actions that should be applied to the system in order to minimise[2] the time required to complete the system's objectives.

The Bellman equation can also be applied to stochastic decision making problems, resulting in an MDP (Bellman 1957*b*). When the state transitions are only partially observable, the policy function cannot be computed exactly, and a POMDP results (Bellman 1957*b*, Papadimitriou & Tsitsiklis 1987, Matthews 2008). When the probabilities are unknown or varying, or the rewards are unknown, POMDPs are often solved by reinforcement learning techniques (Sutton & Barto 1998). All of these problems occur in the TOADM problem, since it deliberately does not exclude the possibility that the state space varies in non-stochastic manner, and all the time components of Equation 6.2 must be expected values rather than exact.

Future work will further explore the similarities between the TOADM problem and these related techniques, and provide a more mathematically rigorous analysis.

6.4.5 Alternative Estimation Techniques

The estimation techniques described in Chapter 5 were shown to be capable of better performance than existing techniques, however the statistical methods used were quite naïve. Although reinforcement learning is likely to be an appropriate class of techniques, there are far more sophisticated methods than the variation on TDL that was used in all the statistical techniques.

Related to this, it may be interesting to compare the TOADM concepts with the well known multi-armed bandit problem (Sutton & Barto 1998). In this problem, a machine with multiple arms offers a reward drawn from a different distribution associated with each arm. In order to maximise one's total reward, the best known strategies (including TDL, Q-Learning, and SA) involve a compromise or switching between exploration and exploitation; testing each arm, and then focussing on the arm with best rewarding distribution.

[2]Minimisation of a cost function is equivalent to maximisation of a reward function.

The statistical techniques employed in Chapter 5 can be considered to perform both exploration and exploitation within a single iteration, equivalent to pulling multiple arms, but only taking the best reward. These strategies could be analysed as a variation of the multi-armed bandit problem in which there is a cost (part of the calculation time, t_C) associated with assessing each arm. The problem is further complicated by the fact that the distributions for each arm may evolve in a non-stochastic manner.

Related work by Sykulski et al. (2010) has explored this problem using adaptive online policies, but only for finite-time problems, and for static distributions on each arm (albeit learnt over time, and thus accounting for the uncertainty). Similarly, Tran-Thanh et al. (2010) has explored the problem in the context of a limited budget, where pulling an arm has an associated cost, and the aim is to maximise the reward given this budget. The concept of a variation of the multi-armed bandit problem in which the arm's distributions vary over time, where the game is unbounded in time, and where the cost to pull and the reward from each arm are also measured in units of time, is certainly unexplored by the mathematics community. Posing such a problem with the TOADM domain may prove an interesting area of further research.

6.4.6 Application Outside the Planning Field

While the class of TOADM problems was shown to be broad, this thesis has only analysed the TOPE problem in detail, and there is scope for much future work in alternative domains. It was suggested in Chapter 1 that there are many possible fields for the application of this work, including autonomous vehicle navigation, GPS navigation software, automated share trading, computer games, search algorithms, and more.

To motivate just one of these possible examples, consider the advent of "millisecond trading" (or "high-frequency trading") within the financial services industry. In this application, electronic trading software makes decisions to buy and sell shares in extremely short time periods. Current systems are difficult to analyse since their developers seldom publish their techniques, but most descriptions suggest they make use of relatively simple condition-action rules to leverage arbitrage techniques when there are short term price differences between markets, or when prices deviate from short or longer term trend lines (Daly 2010). There are suggestions that this strategy will diminish in effectiveness are more such traders

join the market, since its performance is dependent upon being the first to exploit a market opportunity (Urstadt 2009). The use of TOADM techniques could potentially allow more complex types of market opportunities to be exploited, making quick and simple or more deliberated but time-consuming decisions as appropriate.

6.5 Conclusion

This thesis has introduced a novel class of problems known as TOADM. The main contribution was the derivation of a simple equation that is embedded into an existing ADM in order to dynamically manipulate system parameters while the system is in active operation. When applied to the TOPE problem for autonomous vehicles, several techniques were presented that were shown to be capable of significantly reducing the total time to achieve a goal, when compared to any technique that employed only fixed parameter values.

Appendix A

The Hierarchical Replanner

This appendix embeds Allen et al. (2007), *A Planning System for Autonomous Ground Vehicles Operating in Unstructured Dynamic Environments*, in its original form. A PDF version is available from `http://cas.edu.au/download.php/Allen2007ACFRpaper.pdf?id=1843`

A Planning System for Autonomous Ground Vehicles Operating in Unstructured Dynamic Environments

Thomas Allen, James Underwood, Steven Scheding
ARC Centre of Excellence for Autonomous Systems (CAS),
Australian Centre for Field Robotics,
University of Sydney, NSW, Australia
{t.allen, j.underwood, scheding}@acfr.usyd.edu.au

Abstract

This paper describes the design and implementation of a path planning system for an autonomous ground vehicle. The system is designed to be flexible, allowing any planning algorithm to be used and any topology of data to be planned over. It employs a hierarchical separation of two planning modules in conjunction with a vehicle model, to achieve continued vehicle motion while planning and the ability to act as either a deliberative or reactive planner, or a hybrid of both types.

Results from both simulation and field trials are presented, and demonstrate the effectiveness of this architecture on a large outdoor ground vehicle. The contributions of this paper are twofold: a flexible planning system capable of large scale missions for autonomous vehicles; and the use of a vehicle model to determine the requirements for safe operation without slowing the vehicle, and the conditions under which this cannot be achieved.

1 Introduction

Path planning for Autonomous Ground Vehicle (AGV) applications most often occurs in dynamic environments, where the timeliness of goal acquisition is typically a high priority. Despite this, a significant proportion of implementations in the literature make the assumption of a static world while the vehicle is in motion, or stop the vehicle while a new plan is prepared. This paper presents a planning system which neither makes this assumption nor needs to stop while computing a new plan, allowing for long-range plans through dynamic environments to be achieved in a timely manner.

Other approaches in the literature can also achieve this objective, but were deemed unsuitable for the particular implementation described in this paper, since they lacked the ability to determine when they were incapable of planning safely. The main features of the system presented here are that it allows any planning algorithm to be used internally, and offers certain guarantees of vehicle safety if the plans can be computed sufficiently fast. This first feature means that the system can be implemented with equal performance to any other planning algorithm, by using this algorithm internally. The second feature means that the trade-off between the frequency of adjustments to the plan, and the distance along a plan to which a vehicle is committed can be optimised (this is discussed further in sections 3.1 and 3.5).

Given that most real-world scenarios have a vehicle's sensors build a model of the environment incrementally, even when the real world is static, this model is clearly dynamic. As a result, the optimum path to a goal should be re-evaluated as this information is updated. The literature on this problem typically falls into two main approaches: deliberative planners, which replan the entire path as fast as possible (or when the environment model changes); and reactive planners, which react to changes in the environment model by modifying the existing plan. Deliberative planning systems have the benefit that any planning algorithm can be used, whereas reactive planners are typically a self-contained algorithm, but can be hundreds of times faster as they need not replan the entire mission at each step [LaValle, 2006].

In the deliberative replanning category, Zelinsky [1992], Stentz [1994], and others describe a brute-force replanning approach, where a complete plan to the goal is replanned every time the perceived environment changes. Such an approach scales poorly with the distance to the goal, typically $O(n^2)$ for a search optimisation algorithm such as A^* [Hart *et al.*, 1968]. This approach can be improved by using more efficient data representations for cost, such as quad-trees [Zelinsky, 1992; Yahja *et al.*, 1998]. An alternative approach known as anytime planning makes a sub-optimal but feasible initial plan, and then iteratively improves this plan until time runs out [Zilberstein and Russell, 1995]. Typically the process is then repeated in a brute-force manner,

but with the advantage that a plan can be aborted if the cost information changes significantly, and a new feasible plan made available rapidly. Likhachev *et al.* [2003; 2005] further describe a reactive form of the *Anytime* A^* algorithm with promising results.

Reactive planners tend to be more efficient than deliberative replanning methods, even with an optimised data representation and planner, but are typically more complicated to implement [LaValle, 2006]. The D^* algorithm, [Stentz, 1993; 1994; 1995; Stentz and Hebert, 1995] is arguably the best reactive planner for autonomous robotics applications, and operates by propagating changes in cost information throughout the state space of the planner. With these cost propagations in place, a new plan can be generated far more efficiently, as it can reuse the existing plan, accounting for the changed regions which comprise only a subset of the data. Variants of the D^* algorithm such as D^* *Lite* [Koenig and Likhachev, 2002] and *Field* D^* [Ferguson and Stentz, 2006] further improve the efficiency and applicability.

Both classes of planners have been shown to be effective in particular applications, and reactive planners using the D^* algorithm have been implemented in several AGV systems, such as the Mars Rovers [Tompkins *et al.*, 2004; Carsten *et al.*, 2007]. Despite the many successful implementations of such methods, the vast majority only work as well as described for the particular application they solve. Carsten *et al.* [2007] describe such a limitation with regards to large scale mission planning for the Mars Rovers, where the system was suitable for its original task, but ran into failures when this task was extended.

2 Motivation

The main motivation for the planning system described in this paper is the desire to achieve persistent autonomy, which requires a number of features from a planner that are unavailable in current implementations. This paper takes the definition of persistent autonomy as the ability for a vehicle to autonomously achieve goals in an unbounded area, and to determine the conditions under which this ability is compromised. A further motivation is that for many real-world AGV systems, including the Argo vehicle upon which this planning system is implemented, it is important to reach these goals promptly. This means that it is infeasible (or at least undesirable) to stop the vehicle to acquire additional sensor data or while a new plan is being prepared.

An effective path planning system should be reusable by different vehicles, in different environments, and with completely different path optimisation objectives. The system described in this paper achieves these requirements by designing each sub-section of the system in a generic and reusable manner. It separates the sensor data information from the cost information, allowing for different vehicles to plan differently through the same environment. It further separates this cost information from the data structure used to encapsulate it, meaning that an optimised data structure such as the framed quad-trees suggested by Yahja *et al.* [1998] can be used as easily as a simple grid. Finally, it provides a general solution to achieve the combined objectives of traversing an unknown environment along an optimal path, and avoiding obstacles while doing so.

As described in section 1, the assumption of a static environment is untenable for a real world system in which sensors produce a model of the world incrementally. Given that the environment must thus be considered dynamic, this motivates a system which commits to as little of a plan as possible, on the assumption that it may change in the next planning cycle (whether this is reactive or replanning). Many current planning systems which continue to execute while refining their plans – such as those using the D^* or anytime algorithms – solve this problem by allowing both the plans and the vehicle's execution of them to be imperfect, and then correct the errors due to this in the next planning cycle.

This solution is sub-optimal, because it results in errors from the first plan and execution propagating to the subsequent plans. Consider a vehicle with a constant cross-track error, which is slightly off its desired path. Each new plan will instruct this vehicle to move back onto the desired path, but the cross-track error will prevent this from occurring, instead moving the vehicle onwards with this same error. All subsequent plans will continue to request movement towards the desired path and yet never achieve it, propagating this error onwards. If the error is increasing, rather than constant, this problem is continually exacerbated.

The system presented in this paper instead predicts the future position of the vehicle after the estimated planning time for the next cycle, and plans from this point onwards. This vehicle model will necessarily be imprecise, and hence this solution will also be imperfect. However, because the vehicle model is incorporated in the planning system, this system can solve for problems such as the cross-track error example by feeding the error between predicted and measured vehicle behaviour back into the model and controllers.

A final motivating factor for the separation of the planning algorithm from the overall planning system, is that an algorithm can be optimised or replaced, yet the vehicle's behaviour remain constant. In the system presented here, the path plans can be performed by the A^* algorithm just as easily as the D^* algorithm, Vector Force Fields, or even Markov Decision Processes. Regardless of which is used, the type of behaviours exhibited by the vehicle are the same.

3 Design

The motivating factors described above require a general algorithm which can support any type of planner, and one which avoids the problems associated with planning while moving. This section first formulates the minimum reaction time required by the vehicle to avoid obstacles, and illustrates the trade off between this reaction time, and the planning horizon. Next, it presents a design for a replanning system which takes this reaction time into account, and achieves the twin objectives of unknown terrain traversal, and local obstacle avoidance.

3.1 Planning While Moving

All planning takes a finite time – regardless of the algorithm used – in which a vehicle may have moved a significant distance. As a result of this motion, planning systems which do not stop the vehicle while planning must invalidate the first section of a plan before it is sent to the vehicle, lest it turn around to reach the start of this new plan, which it has already passed. This solution is potentially hazardous if the environment changes significantly during the planning time, but is also inefficient because computation time is wasted on a segment of path which is then invalidated.

An alternative solution is to use a vehicle model to predict the future position of the vehicle at the time the plan is completed. The plan can then start from this future position, avoiding the need to invalidate any of the plan. Furthermore, this future position can be explicitly calculated to reflect the dynamics of the vehicle.

Let the vehicle's current velocity, $\dot{x_t}$, be a function of the control inputs, u_t, and the vehicle's current state, x_t:

$$\dot{x_t} = f(u_t, x_t) \tag{1}$$

The future state, $x_{t+\Delta t}$, can then be calculated as the integral of the velocity, over some time interval, Δt:

$$x_{t+\Delta t} = \int_t^{t+\Delta t} \dot{x_t} dt = \int_t^{t+\Delta t} f(u_t, x_t) dt \tag{2}$$

This integral can either be solved numerically, or approximately by a Taylor series, to give the future state of the vehicle for any time increment, Δt.

A number of practical features arise from equation 2. Since it is used to predict the future position of the vehicle after the time required for a plan, the error between this prediction and the real position is directly proportional to the planning time, t_P. Thus, for a fixed state error, a planning system can take a longer time to plan, at the expense of needing a better vehicle model or solution to equation 2. Alternatively, if the planning time is reduced then the quality of the model becomes less important, since the vehicle moves less far in each planning cycle. Dynamic events which cannot be predicted by the vehicle model can thus be reduced to an acceptable level if the planning time can likewise be reduced.

The most significant attribute of equation 2 is a guarantee of vehicle safety. Equation 2 can be used to determine the time, t_S, and distance, s_S, required to stop the vehicle for a given velocity, $\dot{x}$. If an obstacle is detected upon the vehicle's current path at a distance less than s_S, it is impossible to avoid it. Any obstacle further than this distance is safe, because the vehicle can stop before reaching it. To avoid an obstacle without the vehicle needing to reduce its velocity, the obstacle must be further than a distance of

$$s_P = t_P \times \dot{x} \tag{3}$$

where s_P is thus the maximum distance the vehicle can move at its current velocity, during the time taken to plan.

Figure 1 shows the required vehicle actions for obstacles within particular ranges of the vehicle. If

$$t_P \leq t_S \tag{4}$$

then the area of region two reduces to zero, indicating that planning is sufficiently fast to produce plans which avoid all obstacles save those which are completely unavoidable given the vehicle's current velocity.

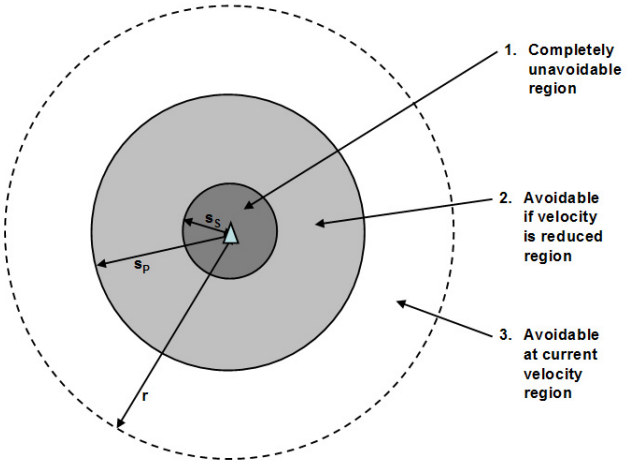


Figure 1: Required vehicle actions for obstacles on the current path within certain ranges.

The following section describes a planning system which makes use of this inequality to guarantee obstacle avoidance within a local region around the vehicle.

3.2 An Abstract Planner Construct

To meet the requirements motivated in the preceding section, this planning system requires an abstract Planner construct to represent the means by which an optimum path is determined between two points. A Planner then uses several other constructs, designed in such

a way that plans can be evaluated over any type and structure of data. This Planner construct is described as abstract because it does not implement a planning algorithm, but instead sets up a series of constructs with which such an algorithm can be implemented. The constructs required to define a Planner are as follows, and their inheritance is shown in the Unified Markup Language (UML) diagram in figure 2.

- A CostSet is a set of cost data, X_C, where each node, $x_i \in X_C$, represents the cost for a plan to pass through the location represented by i[1].
- A ConnectivityFunction defines the topology of a CostSet, X_C, allowing the data structure of the CostSet to be independent of this topology. It takes a given node, $x_i \in X_C$, and returns a list of all n nodes, $\{x_j, x_{j+1}, ..., x_{j+(n-1)}\}$, connected to it.
- A DataSet is a set of arbitrary data, Y_D, in a particular topology, where each node, $y_i \in Y_D$, is a single data point.
- A CostFunction is a function which returns the CostSet element, x_i, whose cost is calculated by applying a function to those nodes in the DataSet, Y_D, which contribute to this element.
- A DataCostFunctionCostSet is a special type of CostSet, X_C, which uses a CostFunction to produce the element, $x_i \in X_C$, whenever it is accessed.
- Finally, a Planner takes a starting node, x_S, and a goal node, x_G, in a CostSet, X_C, and returns the lowest cost path between them as a connected set of nodes, $X_P = \{x_S, ..., x_G\}$.

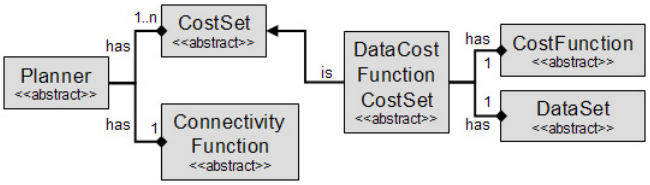


Figure 2: UML Diagram of the Planner construct.

This structure ensures that the data required for a Planner can have any type and structure, provided that the implementation can define a CostSet and ConnectivityFunction appropriately. Where a CostSet is not trivially described by the data itself, a DataCostFunctionCostSet can be used to build a CostSet from a DataSet and a CostFunction. For example, if a vehicle's sensors determine the height of an environment in 1cm squares and store these in a DataSet, a DataCostFunctionCostSet could produce a grid-based CostSet with larger cells by using a CostFunction to take the average height from the DataSet in the area represented by each CostSet cell.

As figure 2 shows, a Planner has a ConnectivityFunction, and also n CostSets. Regardless of how the Planner is implemented, provided that it returns the lowest cost path using this ConnectivityFunction over the sum of the CostSets[2], it can be treated as an independent module and be used in a path planning system as described below.

3.3 Planning in the Local Sensor Region

For typical AGV operation in dynamic environments, new environmental information will be restricted to the region around the vehicle which its sensors can perceive, or a larger region if additional data is being fused from other sources. This sensor region is typically much smaller than the overall area in which the vehicle operates, and can thus be planned over in a significantly faster time.

From figure 1, if this sensor region covers all locations the vehicle can reach within its planning time, that is,

$$r \geq s_P \tag{5}$$

and equation 4 also holds, then planning over this region is sufficient to achieve local obstacle avoidance without needing to slow the vehicle while planning.

3.4 The Replanner System

The design of a Replanner system is now presented, which makes use of the Planner construct described above. Figure 3 shows a UML diagram of this system, and a pseudo-code representation of the algorithm is shown in figure 4.

A Replanner has two Planners, known as the LocalPlanner, and GlobalPlanner, and also asynchronously receives updates to its stored CostMap(s) and vehicle state. It takes a set of n mission goals, $X_G = \{x_1, ..., x_n\}$, of which nodes x_i and x_{i+1} are sent to the GlobalPlanner as the start and goal nodes to plan between, whenever node x_i has been reached by the vehicle. If the global plan is ever invalidated (as described below), the starting node is instead given as x_V, the current vehicle position. Whenever a global plan is valid, the LocalPlanner is run as often as possible. Ideally, this should be as fast or faster than new cost data is received, so that each plan is still optimal with respect to the current cost data at

[1]This representation differs from the typical cost graph, where costs are associated with the connections between nodes, rather than the nodes themselves. A CostSet is equally general, and may be a cost graph, or any other topology of information.

[2]Where the overall cost cannot be expressed as a weighted sum of the costs from several CostSets, it can instead be implemented as a single DataCostFunctionCostSet, using an appropriate CostFunction and DataSet which combine the data from all the sources.

```
Main Loop
    while( missionGoals ≠ ∅ )
        get currentVehiclePosition
        if( currentVehiclePosition is within tolerance of currentMissionGoal )
            remove currentMissionGoal from missionGoals
        if( globalPath = ∅ )
            find globalPath from currentVehiclePosition to currentMissionGoal
            if( successful )
                vehiclePath = globalPath
            else
                return FAILURE
        else
            if( dataStore has changed, or currentVehiclePosition has changed )
                get predictedVehiclePosition after averageLocalPlanningTime
                set localPlanningStart to the closest point in globalPath after predictedVehiclePosition
                set localPlanningGoal to the closest point in globalPath before localPlanRange
                find localPath from localPlanningStart to localPlanningGoal
                adjust averageLocalPlanningTime by measuredLocalPlanningTime
                if( successful )
                    vehiclePath = localPath
                else
                    clear vehiclePath, stopping vehicle
                    globalPath = ∅

Update(CostInformation)
    update dataStore with CostInformation

Update(NavigationInformation)
    update currentVehiclePosition with NavigationInformation
```

Figure 4: Pseudo-code presentation of the of the complete Replanner system. The main loop and the update functions run asynchronously (with appropriate data locking around the cost and navigation information).

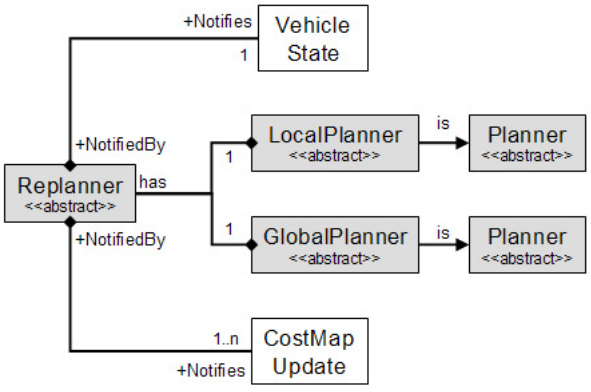


Figure 3: UML Diagram of the Replanner construct.

the time it is completed. A local plan can fail if either the goal point lies behind an impassable obstacle (given the current cost data), or because the Planner it uses will abort a plan upon some condition, such as a time-limit. If this occurs, the vehicle is safely brought to a stop and the global plan is invalidated. If equation 4 holds, this is the only situation in which the Replanning system need ever stop the vehicle while planning.

The Replanner uses the vehicle dynamics model described above to predict the vehicle's position once the next local plan has been computed, using an estimate of this planning time (such as a rolling average of how long previous local plans have taken). The LocalPlanner is then given a starting node that is the next point on the global plan from this estimated vehicle position. The goal node is the furthest point on the global plan that is closer than a specified local planning distance, and which is not an obstacle. This choice of starting and goal nodes is made such that the LocalPlanner is always following the global plan, but locally adjusting it as new information is received. Since the starting position reflects the dynamics of the vehicle, this method explicitly allows the vehicle to continue moving while planning, and avoids the need to invalidate any parts of the plan.

The design presented here is more generic than the implementation described below, as it allows for the LocalPlanner and GlobalPlanner to be separate Planner constructs. However, since the Replanner is guaranteed to never be using both the LocalPlanner and GlobalPlan-

ner simultaneously, it is possible for both of these to use the same Planner and CostSet, as in the implementation on the Argo vehicle. Alternatively, this design allows for the GlobalPlanner to be a much coarser data representation, or even use entirely different data (for example, using a satellite map of an area while the LocalPlanner uses onboard sensors).

3.5 Trade-Offs

As described previously, one of the main benefits of this Replanner design is the ability to determine the effects of adjusting parameters in the system. Defining the frequency at which the cost maps can be updated as f_C, and the frequency at which planning can be achieved as f_P, it is desirable that

$$f_P \geq f_C \tag{6}$$

in order that the system does not compute a plan that is already sub-optimal with respect to the latest cost information by the time it is completed. Equally, it is desirable that equation 4 is upheld, to ensure that all avoidable obstacles can be planned around without the need to slow the vehicle. Equation 4 can also be expressed in terms of distances:

$$s_P \leq s_S \tag{7}$$

The stopping distance, s_S, is a function of the vehicle velocity, $\dot{x}$, and from equation 3, the planning distance, s_P, is likewise. As a result, it is clear that adjusting the vehicle velocity can have no effect on the inequality of equations 4 or 7.

The only other variable that can be adjusted to achieve equation 4 is the planning time, t_P, and clearly

$$t_P = \frac{1}{f_P} \tag{8}$$

by definition. This planning frequency is a function of the computational speed of the planning algorithm, the cost data resolution, the length of the local plan, and potentially other factors. The planning algorithm cannot be optimised past a certain point, so this factor cannot be adjusted. The resolution of the cost data can be reduced to increase the planning frequency, but at a cost to the quality of the plans. Additionally, many implementations would be able to produce cost maps at a faster rate if the data resolution was reduced, meaning that this parameter may lie on both sides of the inequality in equation 6. This leaves the length of the local plan as the only parameter which can directly contribute to the the trade-offs described by equations 4 and 6.

Adjusting this parameter allows the Replanner system to operate anywhere along the scale between reactive and deliberative planners. If sufficient computational time is available, then the local plan length can be the entire length of the global plan, causing the system to act like a brute-force deliberative planner. Alternatively, if the minimum local plan length is just greater than the stopping distance (so that the vehicle is never able to drive past the end of a plan into potentially hazardous areas), the system behaves like a purely reactive planner.

Since the local plans find a path to the furthest point along the global plan allowed by their maximum length, the local plans are better if this parameter is longer, since they can take paths past larger regions of the now sub-optimal global plan, as shown in figure 5. The competing desires to see the local plans both as long and as frequent as possible allows this parameter to be optimised precisely. For optimal planning performance, the local plan length should be the largest distance possible to still allow for equation 6 to hold. The implementation of the Replanner system upon the Argo vehicle sets the local plan length in this manner, and is described in the following section.

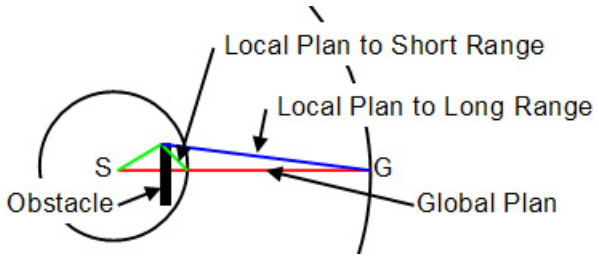


Figure 5: Local plans around an obstacle for a short maximum plan length (green), and a longer range (blue), illustrating that the longer the local plan length, the more direct the planned route.

4 Implementation

The Replanner design was written in a C++ framework, and a particular implementation was then created for the Argo vehicle, shown in figure 6. C++ was chosen primarily because the Planner and Replanner constructs can be easily represented as abstract classes in a object-oriented language. Implementing a specific instance of a Replanner is then a matter of inheriting from these abstract base classes and following the required interfaces. For Argo vehicle application, the A^* algorithm was used to implement a Planner, and was chosen for the simplicity of its implementation, and the provable optimality of its plans (with respect to the cost data) [Pearl, 1984]. A two-dimensional grid topology, implemented as a quadtree, was used for the CostSet, since the current requirements of the Argo vehicle do not require a more versatile representation.

Figure 7 illustrates the overall implementation of the A^* algorithm as a Planner, showing the abstract classes

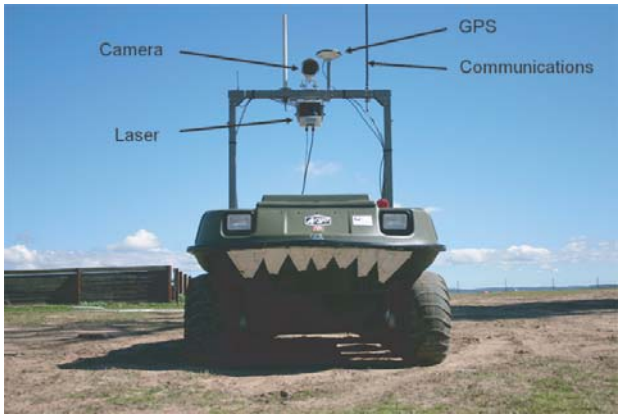


Figure 6: The Argo vehicle, showing sensors and communications hardware.

in light grey, and the specific implementations in dark grey. This figure shows that the implementation uses a two-dimensional grid for a DataSet, which itself uses an efficient C++ quad-tree implementation so that uninitialised data points require no memory for storage. The cost information is constructed from this grid using a CostFunction which multiplies the data value of each node (described below) by the Euclidean distance to the goal node, with an adjustable weighting. The CostSet built from this data has each node connected by an eight-way grid function, (allowing a range of moves equivalent to a king in a game of chess).

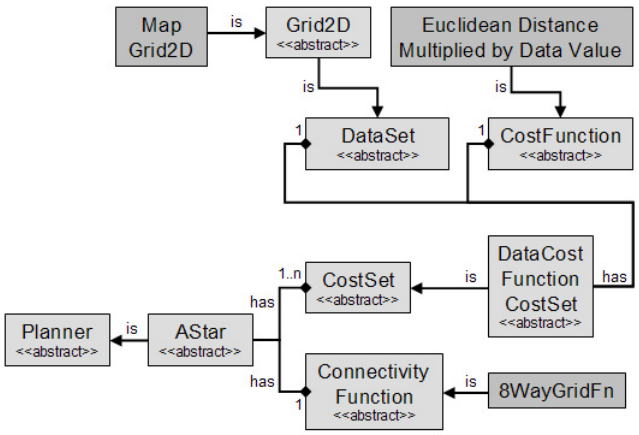


Figure 7: UML diagram showing the overall implementation of the A^* algorithm as a Planner.

As implemented upon the Argo vehicle, the DataSet is built incrementally from the laser data points acquired by the vehicle's sensors. This is built into a terrain cost grid by assessing areas which have either steep gradients between neighbouring grid cells, or a large range of heights within a single cell. These thresholds for terrain steepness and untraversable heights depend on the particular vehicle used; for a different class of vehicle an entirely different method may be more appropriate. Each cost cell in the grid is then expanded by an amount proportional to the vehicle's dimensions, in a process functionally equivalent to a Configuration-Space transform, [Russell and Norvig, 2003; LaValle, 2006].

The Planner implementation allows for any of the n CostSets to veto a node, marking it as untraversable. For example, the Argo vehicle's implementation uses this feature to mark the boundaries of the testing facility, which are otherwise undetectable by the vehicle's sensors. This process is equivalent to making the cost prohibitive, but the A^* algorithm implementation then completely removes this node from the search space, which is a more efficient solution as it reduces the search time by focussing only within the traversable region. The final output of the Planner is an ordered set of nodes, connected by the moves allowed by the ConnectivityFunction.

The Argo vehicle carries three separate computers to manage vehicle control, data processing, and the replanning system, each of which comprises a PC104 stack with a single-core Pentium III processor. With a grid cell size of 30cm, local plans to 11m ahead of the vehicle (the point where the laser scanner hits the ground) can be performed at greater than 20Hz. This is faster than the vehicle can generate an updated CostMap of this region, and far faster than required for equation 4 to hold true. As a result, this implementation is sufficiently real-time for the current requirements of the Argo vehicle.

Finally, this implementation has a number of variable parameters (separate to those in the Replanner system, described in section 3.5) which directly affect the behaviour of the system and the vehicle. Uninitialised data in the CostMap is given a fixed cost which affects the vehicle's tendency to plan through unknown terrain. If this cost is low, then the system will preference paths through unknown areas rather than paths over costly terrain, and vice versa. To allow multiple CostSets to be planned over simultaneously, the Planner construct assumes that their costs can be added in a weighted sum (however, if this is not possible, they can be combined into a single DataCostFunctionCostSet in whatever combination is appropriate). The weighting of each CostSet in this sum can thus be adjusted to reflect the importance of avoiding areas of high cost in each set with respect to the others. The last main parameter is the weighting of plan length versus cost, resulting in a preference for longer paths which travel over less costly terrain if it is raised, and vice versa if it is lowered.

5 Results

Results of this Replanner implementation are now presented, firstly in simulation, and secondly running in real-time on the vehicle, with no changes to the implementation. The simulated results used an accurate vehi-

cle model and simulated drive-train, and read in cost information from a pre-generated image in an area around the vehicle equivalent to the sensor field of view, at the same rate as a CostMap could be generated by the vehicle. The simulation was run both with cost information obtained from a helicopter fly-over of the test area, where the helicopter carried the same SICK laser scanner as the Argo vehicle, and also with data directly from the vehicle driving through the same area. The simulated vehicle exhibited similar behaviour with both sources of cost information. The real-time results used exactly the same implementation, save that the CostMap was generated in real-time from the vehicle's laser scanner.

Figure 8 shows a time sequence of the Replanner system in operation, using the simulated Argo vehicle, and pre-generated cost information. In the figures, the background image is a high resolution photo of the test area, and the cost information is incrementally overlayed upon it as the vehicle moves. The pink circle shows the current mission goal, and the red circles show the current global plan to this goal. The green circles are the current local plan, and the blue circles are the path to which the vehicle is committed, (which in these images is a fixed length because it only reflects the vehicle's maximum stopping distance, since the planning time was constant).

Figure 8-1 shows the initial global plan, which in the absence of cost data, is merely a straight line to the goal. With no obstacles seen in the local region, the green local plan simply follows the global plan. Figure 8-2 shows the green local plan diverging from the global plan to take a better route, given the new information just acquired by the sensors. This frame also shows that the global plan crosses an obstacle which has just been discovered by the vehicle's sensors. This has invalidated the global plan, stopping the vehicle (hence the blue committed path is not seen). Figure 8-3 then shows the new global plan which avoids this obstacle, from the vehicle's current position to the goal. Finally, figure 8-4 shows the vehicle's path with small red circles, and the total cost information gathered by the vehicle during this mission.

The attached video [3] shows the same Replanner implementation, running in real-time on the Argo vehicle. The vehicle was placed in an area of the test facility with no initial environment information, and was instructed to drive approximately 150m to the far side of a thicket of dense scrub. The video shows the vehicle performing this mission, illustrating the vehicle needing to redo its global plan and back-track at one point. A screen capture of the user interface software illustrates its route midway through the mission. The same mission (given by three waypoints) was then repeated using the helicopter fly-over as initial cost data. With this initial data, the vehicle calculated a far better path (with respect to its CostFunction), first driving away from the goal to reach a flat road, then proceeding around the entire scrub area. The last screen capture demonstrates the faster and less complicated path performed in this second mission. In both cases the vehicle reached the goal in an appropriate manner, with the Replanner system operating as expected. Subsequent missions with improved communications hardware on the Argo vehicle have used the same Replanner system over distances of around 2km with equal success.

[3] Available at `http://www-personal.acfr.usyd.edu.au/t.allen/ACRA2007/ReplannerResults.mpg`

6 Discussion and Future Work

One criticism of this Replanner approach is that predicting the future position of the vehicle and planning from this point is functionally equivalent to a reactive planner which allows imperfect execution of its plans, and then corrects for this in the next cycle. This is correct, but neglects the advantages of the Replanner system, in that its makes the trade-offs involved explicit and adjustable. A reactive planner such as the D^* algorithm is forced to account for all updated cost data at each planning cycle, whereas the Replanner approach can potentially plan over only a smaller subsection of this changed data. Even if the reactive planner can plan through this updated data in an efficient manner, depending on the quantity of data involved, this can still be a computationally expensive process. An anytime algorithm can partially solve this problem by aborting the plans earlier, but sacrifices their optimality to do so. By comparison, the Replanner approach can continue to produce optimal local plans (with respect to the current cost data) at the required rate, and the only trade-off is to limit how much they can improve upon the global plan because they must follow it more closely.

This Replanner system has been shown to be practical and efficient in its implementation on the Argo vehicle, yet there are several areas in which improvements could be made, and where further research is likely. Even with a planning system which generates the optimum path with respect to the cost information, it is still difficult to accurately control a vehicle along it. There are many solutions to this problem, such as using a non-holonomic planner which can reject paths that are impossible for the vehicle to follow, or applying a path optimisation stage to the algorithm to smooth or otherwise modify the path. The state lattices proposed by Kelly *et al.* [Pivtoraiko and Kelly, 2005; Knepper and Kelly, 2006] are an effective such method, and can be readily implemented in the Replanner framework. For the skid-steered Argo vehicle, a wheel-slip estimator has been designed, which should allow the vehicle to better control its steering at all speeds, resulting in better path following.

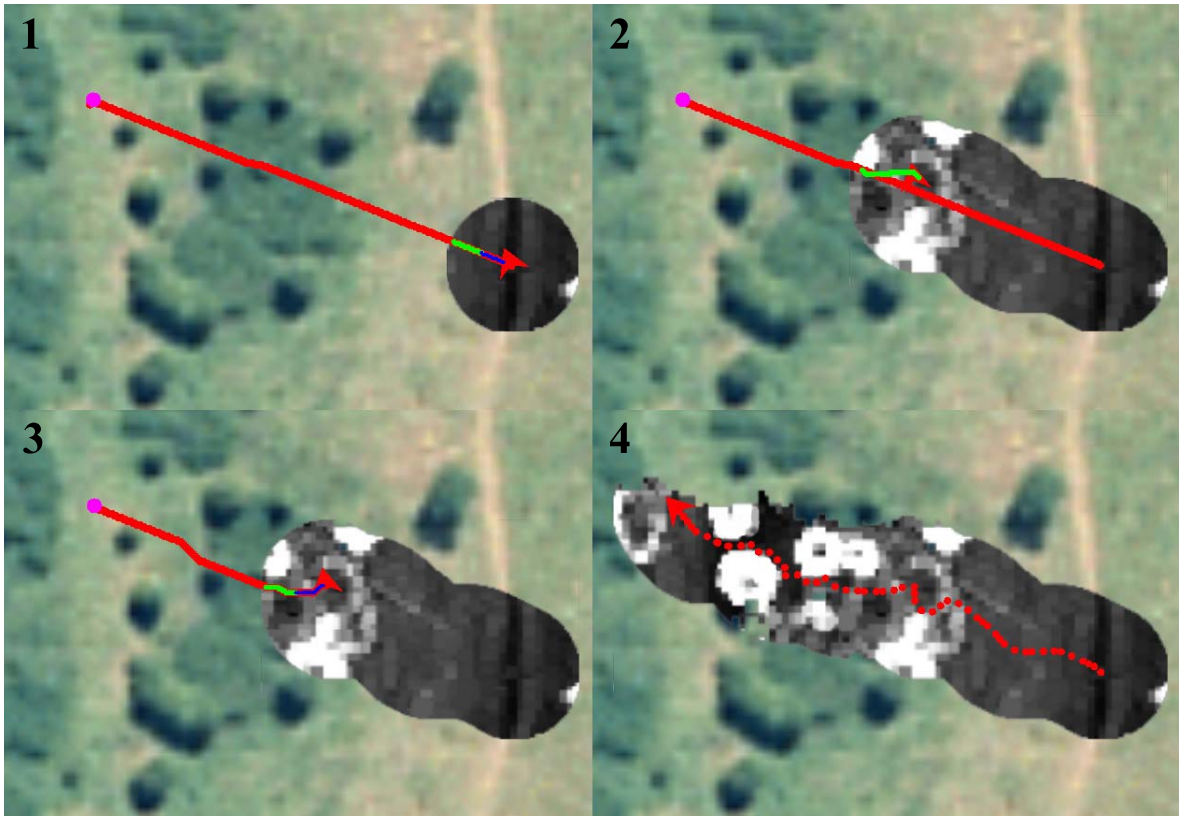


Figure 8: The Replanner algorithm in simulation.

All discussion in this paper has described the paths produced as optimal with respect to the cost information, and not necessarily optimal if this information is wrong or insufficient. It is arguable however that if the cost information is deficient, then subsequent problems are the result of this, and not the planning system used. The Replanner system presented here removes many of the difficulties associated with planning, while moving them to the area of cost map generation and vehicle control. In the Argo implementation, these cost maps are able to be generated in real-time at approximately 10Hz, sufficiently accurately to allow for optimal planning. If this was not the case, then the results obtained – while still correct with respect to the cost information – would not be as satisfactory as presented here. For the Argo implementation, the largest computation effort for cost map generation is expanding the obstacles by the vehicle's dimensions. It is believed that this task could be equivalently performed using an efficient image dilation technique directly on a consumer Graphical Processing Unit (GPU), which would free up CPU time and allow the vehicle to plan even faster.

Future work in this area will focus on two main objectives: the fusion of additional data from other vehicles and the subsequent use of the Replanner system for multiple vehicle control; and testing the system with truly dynamic obstacles such as other vehicles, livestock, or pedestrians. It is believed that the Replanner system is general enough to handle such situations, but only if the cost information can represent the changing environment sufficiently quickly and accurately, and the vehicle can be controlled effectively along the rapidly changing paths.

7 Conclusion

This paper has presented the Replanner system for AGV navigation in unstructured dynamic environments. The system has the ability to operate anywhere along the scale between reactive and deliberative planners, and the optimum point on this scale can be determined for any given implementation. Results of this system as implemented on a large ground vehicle known as the Argo were presented, and show that the system is suitable for achieving persistent autonomy. Future work will focus on producing cost maps in a manner to allow dynamic obstacle avoidance, and the use of the system for multiple vehicle control.

The main contributions of this paper are twofold: a framework for a flexible planning system to achieve persistent autonomy, capable of using any planning algo-

rithm, and any type and structure of cost data; and the use of a vehicle model in the planning system, which allows the requirements for safe operation to be determined, and without the need to slow or stop the vehicle while planning.

References

[Carsten *et al.*, 2007] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz. Global path planning on board the mars exploration rovers. In *IEEE Aerospace Conference*, 2007.

[Ferguson and Stentz, 2006] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2 2006.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 2:100–107, 1968.

[Knepper and Kelly, 2006] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[Koenig and Likhachev, 2002] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *International Conference on Robotics and Automation*, 2002.

[LaValle, 2006] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available from `http://planning.cs.uiuc.edu/`.

[Likhachev *et al.*, 2003] M. Likhachev, G. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Neural Information Processing Systems*, 2003.

[Likhachev *et al.*, 2005] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling*, 2005.

[Pearl, 1984] J. Pearl. *Heuristics*. Addison-Wesley, 1984.

[Pivtoraiko and Kelly, 2005] M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search state lattices. In *The 8th International Symposium on Artifical Intelligence, Robotics, and Automation in Space*, 2005.

[Russell and Norvig, 2003] S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 2003.

[Stentz and Hebert, 1995] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. In *International Conference on Intelligent Robots and Systems*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1995.

[Stentz, 1993] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical report CMU-RI-TR-93-20, Carnegie Mellon Robotics Institute, 1993.

[Stentz, 1994] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*, pages 3310–3317, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1994.

[Stentz, 1995] A. Stentz. The focussed d* algorithm for real-time replanning. In *International Joint Conferences on Artificial Intelligence*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1995.

[Tompkins *et al.*, 2004] P. Tompkins, A. Stentz, and D. Wettergreen. Global path planning for mars rover exploration. In *IEEE Aerospace Conference*, 2004.

[Yahja *et al.*, 1998] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *IEEE Conference on Robotics and Automation*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1998.

[Zelinsky, 1992] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6), 12 1992.

[Zilberstein and Russell, 1995] S. Zilberstein and S. Russell. *Imprecise and Approximate Computation*, chapter Approximate Reasoning using Anytime Algorithms. Kluwer Academic Publishers, 1995.

Appendix B

The Parallel Hierarchical Replanner

This appendix embeds Allen et al. (2009), *Dynamic Path Planning with Multi-Agent Data Fusion - The Parallel Hierarchical Replanner*, in its original form. A PDF version is available from `http://cas.edu.au/download.php/Allen2009ICRADynamicPath.pdf?id=2042`

Dynamic Path Planning with Multi-Agent Data Fusion - The Parallel Hierarchical Replanner

Thomas Allen, Andrew Hill, James Underwood, and Steven Scheding

***Abstract*— The design of a hierarchical planning system in which each level operates in parallel and communicates asynchronously is presented. It is shown that this Parallel Hierarchical Replanner is both reactive, and as close to optimal over all information in the state space as is possible given finite computational power. A comparison with three other hierarchical methods is presented, which demonstrates that for scenarios in which the time taken to achieve a mission goal is of greater importance than the cost incurred, this approach has better performance than related methods in the literature.**

I. Introduction

Many planning systems in the literature can be differentiated by their use of either deliberative or reactive techniques. In the deliberative case, all available information from the past and present is used to determine a complete plan to achieve a goal. In the reactive case, only the latest information is considered, typically via a set of condition-action rules to produce a 1-step plan [1]. While reactive techniques are much faster in general than typical deliberative methods, they can produce results that are far from globally optimal.

Much work has focussed on attempting to fuse the speed of reactive and the optimality of deliberative methods into a hybrid system. This is often achieved using a hierarchical system consisting of a reactive lowest level planner, and a deliberative highest level. Examples of such systems are found in many different areas of robotics, including the RoboCup soccer competitions [2], outdoor Autonomous Ground Vehicle (AGV) navigation [3][4][5], and interplanetary exploration [6][7].

One area, which has not been as thoroughly investigated, is parallelisation of the planning process. This is motivated by the reduced cost and increasing prevalence of multi-core processors, and the hypothesis that such an approach will allow for near globally optimal results at speeds comparable to reactive methods. Although many reactive and even some efficient deliberative methods are fast enough to render this unnecessary for single agent applications, in most cases however, these same methods are unsuited to the multi-agent data fusion scenario. The D^* algorithm for instance is optimised for repairing plans due to changed information acquired close to the agent's current location, an assumption which is violated when multiple agents can acquire information anywhere in the planning state space.

All authors are with the Australian Centre for Field Robotics, University of Sydney, Australia. {t.allen, a.hill, j.underwood, scheding}@acfr.usyd.edu.au

This work is supported in part by the Australian Research Council (ARC) Centre of Excellence programme, funded by the ARC and the New South Wales State Government. Additional support is provided by the Australian Defence Science and Technology Organisation.

A parallelised hierarchical planning system is shown to be better suited to this problem, as the different levels of planners account for both local and remote exteroceptive sensor data simultaneously. Furthermore, the hierarchical model is inherently parallelisable as the separate levels can be computed independently, provided that the interface is designed appropriately. This paper motivates and then describes extensions to the existing Hierarchical Replanner (HR) presented in [5]. These improvements are validated by comparative simulation with other methods, and also via a field demonstration involving multiple AGVs operating cooperatively to traverse an unstructured outdoor environment. This demonstration was part of the Centre for Autonomous Systems' Outdoor Research Demonstrator (CORD) program at the Australian Centre for Field Robotics.

II. Motivation

Given unlimited computational power, the intuitive solution to the the planning problem in dynamic environments is to treat it as a continuous optimisation over some cost, and to use a brute-force approach to repeatedly compute and execute the optimum plan. For a real-world system there are limits to computational power, and the best solution is that which gives the best approximation to the continuous optimisation. Henceforth in this paper, 'local' and 'global' planners are used to refer to the lowest and highest levels respectively in a multi-level hierarchical system, and a 'globally optimal' plan is one which is optimal over all information in the state space of the global planner.

What constitutes the best approximation to the continuous optimisation is application specific, and depends upon the variables being optimised. For some scenarios it is better to act deliberatively and execute closer to globally optimal plans, and for others to achieve the goal faster but via only a locally optimal approach. To evaluate system performance, any given scenario will have a metric, M, which is some function of n variables, v_i, with associated weights, k_i.

$$M = f\left(k_1 \cdot v_1, \ldots, k_i \cdot v_i, \ldots, k_n \cdot v_n\right) \tag{1}$$

The best approach is that which optimises this metric for the particular set of weights and variables relevant to the scenario at hand.

Prior work in [5] showed that hybrid planning systems feature a trade-off between the reaction time required to repair a plan given new information, and the cost of this plan compared to the global optimum. In the HR system described therein this trade-off was an adjustable parameter, and was balanced at a point which guaranteed vehicle safety.

This guarantee was achieved in the two-level hierarchy by increasing the local plan lengths until the maximum time required to compute a plan equalled the maximum time required to stop the vehicle. Increasing the local plan lengths leads to plans which are closer to the global optimum, but take longer to compute and thus increase the reaction time. The choice to prioritise vehicle safety over other factors thus sets an upper bound on how close the path plans can be to the global optimum.

The primary motivation for this work is to achieve closer to globally optimal paths, but with reaction times comparable to reactive methods. To define what is meant by reactivity, it is argued that whether explicitly stated or not, every autonomous system features a bound on how far the agent is permitted to traverse through stale sensory data. In many implementations this bound is the distance travelled in the time taken to process a batch of sensory information, whereas in others it is related to the dynamics of the agent. Even in the near instantaneous condition-action rule approach of some reactive systems there is a finite evaluation time involved, during which the agent must still be moving through information known to be stale. If the scenario dictates that the system should not permit travel through stale data and stops the agent when this would occur, then the agent's inertia will still carry it some finite distance. Whether or not a planning system accounts for this feature, it is present in any physical system and thus motivates a description of reactivity which accounts for it.

The following equation defines the conditions required for a planner to be 'sufficiently reactive', whereby the maximum time taken to compute a plan, t_p, is less than the minimum time taken to travel this permitted distance through stale data, t_s, given the agent's dynamics.

$$max(t_p) \leq min(t_s) \tag{2}$$

To achieve closer to globally optimal plans requires improvements to the highest level of a multi-level hierarchy for which (2) does not already apply, because lower levels are already sufficiently reactive. If (2) is valid over all available information for the global planner, then the resulting system is globally optimal. Kelly *et al.* [8] describe the PerceptOR program, whose planning system uses the D^* algorithm to achieve this for their global level, but only under certain conditions, unsuited to the multi-agent data fusion scenario.

The secondary motivation for this work is to provide a system which is suited to the multi-agent data fusion scenario, whereby new information can become available anywhere in the state space of the planner. It is argued that any system which relies on a backwards incremental planner (such as the D^* algorithm) is unsuited to the multi-agent data fusion scenario, because the computation time required to repair a plan increases significantly as information is obtained closer to the goal. Intuitively, this is because a backwards planner expands the search outwards from the goal. The closer to the goal that information changes, the greater the number of possible paths that are affected. Thus, the longer an incremental repair will take, since it must reassess many of the nodes on these paths.

Fig. 1 illustrates the results of an experiment designed to verify this for the D^* algorithm, and clearly shows the deterioration in performance. That the multi-agent data fusion scenario frequently exhibits the conditions that cause this problem motivates the design of a planning system which alleviates it.

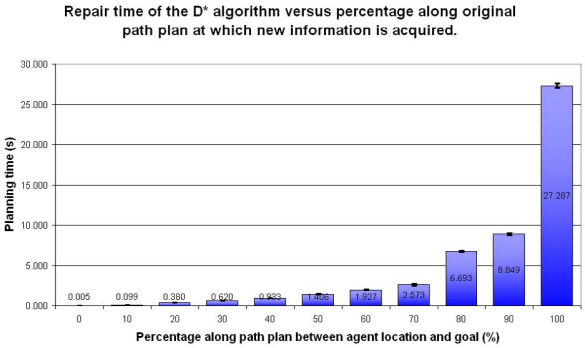


Fig. 1. Results of an experiment measuring the time taken for the D^* algorithm to repair a pre-existing initial path, given a fixed size set of changed information at various distances along the original path, with the error bars showing the 95% confidence interval. The repair time exhibits a significant increase as the changed information is acquired closer to the goal point.

III. Related Work

Hybrid architectures for planning have existed for over two decades, and illustrate a wide variety of interactions between their components. Many authors cite Brooks' subsumption architecture [9] as one of the earliest such methods, which allowed each level to subsume the output of lower ones and influence the higher level responses. This architecture was not specifically designed for planning, rather for integrating all aspects relevant to controlling a mobile agent, however the central idea – that higher levels should be able to read outputs from and write inputs to lower levels through a low-bandwidth interface – is present in many other systems in the literature. Brooks briefly notes that each level could be implemented on a separate processor given such an interface, but does not apply this directly to planning.

Blythe and Scott-Reilly [10] describe integrating reactive and deliberative planning for a household robot, using components known as Hap and PRODIGY respectively. The most interesting aspect of this hybrid system is that while Hap has a list of possible condition-action rules like many reactive planners, invoking PRODIGY to do a new deliberative plan is simply one of these actions, conditioned on a low demand for any other reactive need. The scope of the deliberative plans is limited by the reactive level's estimate of how much time is available before another reactive need takes precedence.

Jensen and Veloso [2] describe the interleaving of deliberative and reactive planning for dynamic multi-agent domains in the context of the RoboCup soccer challenge.

Their approach requires that the state space of the deliberative planner remain constant while planning (or else it is recomputed accounting for these changes), and tries to meet this assumption by discretising this space as a function of the average planning time required. This does not guarantee that the resulting plan is directly useable, but with a suitable discretisation function, a low probability of needing to recompute can be obtained.

Kluge and Morgenthaler [3] describe a part of the PerceptOR program known as the *Raptor* system, which fuses three different planning methods into a 'command-arbitration' architecture. A reactive trajectory generator produces steering arcs in response to obstacle proximity, elevation, and slope changes. Next a Rapidly-exploring Random Tree planner uses local terrain data to continuously find paths that meet the intermediate waypoints produced by a global planner which uses the D^* algorithm. An arbitration step performs a weighted sum of the steering commands produced by the two lowest levels, using the argument that each level is both reactive and suboptimal, and that this sum represents the best combination of both approaches. The global planner has a bounded state space to ensure its plans have an upper limit on computation time.

Kelly *et al.* [8] describe the full culmination of the PerceptOR program, which builds on the *Raptor* system. The planning aspects of this program are hierarchical and hybrid, and also use the D^* algorithm (the *Field* D^* variant [4]) for global planning in much the same manner. Only one lower level is present; a local planner known as the *Ranger* which uses a forward model to predict the agent's trajectory along a discrete set of velocity and steering commands in the space of control inputs [11]. Possible trajectories are eliminated if they would cause the vehicle to collide with an obstacle, and the remaining set are matched to the closest possible starting cells in a grid representation, with a penalty for slight misalignments. A field interface is used to allow the local planner access to the global D^* algorithm itself, making the overall path the lowest cost combination of local and global path sections.

The PerceptOR approach produces a set of feasible rather than optimal local plans, yet the system still achieves globally optimal performance. This is possible because the system computes the optimal combination of one of these local plans with the global plan, making the overall path globally optimal. Where the PerceptOR approach breaks down however is in the implicit coupling between local and global levels. The global level is required to replan at every iteration of the local, and the approach is only successful where global planning is sufficiently reactive. This is a trade-off to allow the Ranger planner to model non-holonomic vehicle constraints, which often result in local plans which cannot follow the globally optimal plan. Without this coupling, non-holonomic planning would not be possible in the PerceptOR program, yet with it in place, the system is only suited to certain constrained multi-agent scenarios.

Although the PerceptOR program performed multi-agent trials, their aerial vehicle was flown directly above the ground vehicle [8]. Such a configuration ensures that the aerial vehicle's information is acquired close to the starting point of each global plan on the ground vehicle, allowing repairs to be computed quickly. As Fig. 1 illustrated, the D^* algorithm's performance deteriorates with the distance of new information from the starting point, and hence it is argued that the PerceptOR approach is not suited to data fusion from unconstrained configurations of multiple agents.

Discussion of the choice between following a known suboptimal plan and waiting while a repair is computed is an important topic, which appears to be inadequately covered in the literature. All of the systems described above avoid making this choice; the PerceptOR program is sufficiently reactive in its chosen scenarios that the choice is irrelevant, Jensen and Veloso's approach assumes the state space remains constant during global planning, and both Brook's subsumption architecture and Blythe and Scott-Reilly's approach wrap each level inside another so the agent only follows the final output.

As Fig. 2 illustrates, either choice can be poor in certain situations, especially if a particular observation has the potential to dramatically alter the plan. The best option depends on the weightings for total mission execution time, t_m, and the total cost of execution, c_m, in the overall metric. For the CORD program, the cost being optimised is a representation of traversability, and hence the cost of the path produced directly determines t_m since the CORD AGVs drive slower over rougher terrain.

IV. The Parallel Hierarchical Replanner

This section describes the Parallel Hierarchical Replanner (PHR), in which each level of the hierarchy operates independently. The requirements for a planning system suited to the CORD program are first presented, followed by a description of the PHR framework design, and the particular implementation on the CORD AGVs.

A. Requirements

Given the motivations presented in section II, the requirements for such a system are as follows:

1) Each level of the hierarchy must inform the system when it fails.
2) Each level must support any type of planning algorithm, with the inputs and outputs expressed in a common language.
3) The hierarchy must support all combinations of one or more levels.
4) At least the lowest level of the hierarchy must be sufficiently reactive.

Item one reflects the desire to achieve persistent autonomy; the ability for a vehicle to autonomously achieve goals in an unbounded space and timespan, and to determine the conditions under which this is unachievable. By informing the system when a level fails, these conditions become known, and the relevant actions can be taken.

Item two reflects the desire for flexibility in the system. Of particular relevance is the fact that various types of

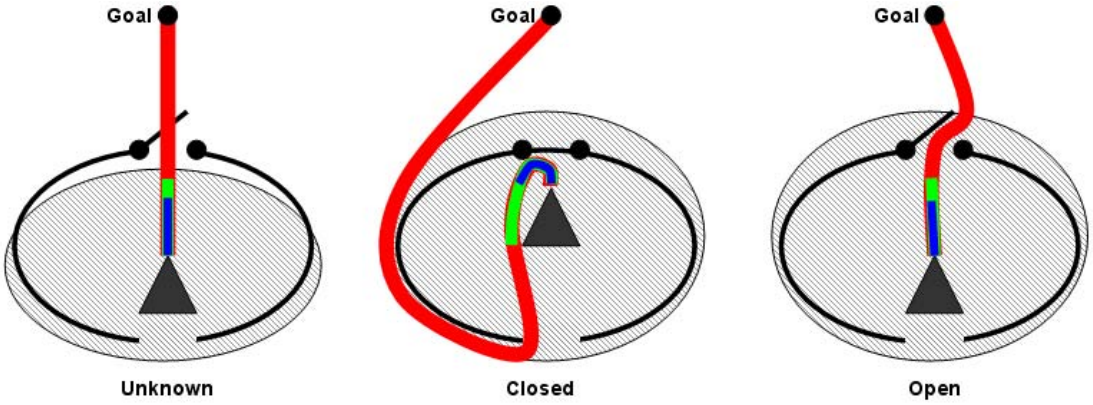


Fig. 2. In the left image, the agent is guided by the global plan (shown in red) into a cul-de-sac, because the observed region (shaded in grey) does not yet include the gate at the end. Once this gate is perceived and a global plan repair commences, there are two possible choices while it is computed; either the agent follows the committed path (shown in blue) or it remains stationary. If the gate is closed as in the centre image, then following the committed path increases c_m because the agent must now backtrack from its new position. Alternatively, if the gate is open as in the right image, then remaining stationary increases t_m because the changes to the global plan were minimal and the agent could have continued.

planning algorithms are suited to different tasks because of their different computational properties. As a result the appropriate planning algorithm should be selected for each level in the hierarchy, and possibly for each different task or environment. The requirement of a common language both simplifies the design and implementation, and allows the potential for communications between levels to be redirected in a more complex framework.

Item three is required for complete generality; if the system is sufficient with only one level then there is no reason to enforce a larger hierarchy.

A system that meets these first three requirements is a superset of all the systems described in section III; it can be used to build Brooks' subsumption architecture, the PerceptOR system, and the PHR framework described below. Item four is relevant to any scenario in which the time taken to achieve a goal is significant. Provided that the lowest level is sufficiently reactive, and the plans are longer than the agent's stopping distance, then a new path can be computed to avoid obstacles or stop the agent before it reaches them. If this can be guaranteed, and the agent's sensors are sufficient to perceive this immediate environment, then it is safe to allow the agent to continue execution of a lower level plan while higher levels are being computed.

B. Design

The design of the PHR builds on the two-level HR presented in [5]. Briefly, the HR is encapsulated in a 'Replanner' structure which implements the logic required to compute local and global plans. This structure holds two separate 'Planners' which both have access to the asynchronously updated 'CostMap' containing information over which to plan. The Replanner runs as a single threaded function which continually computes local plans provided a global plan is available, and only recomputes the global plan if a local plan fails or cannot guide the agent back onto the global plan.

The goal point for each local plan is the furthest point along the global plan within a specified distance of the agent's position. As described in [5], this distance is determined as a function of the sensor range and dynamics of the agent, and the the upper bound on computational time for the local planning algorithm, in a manner which then guarantees that the agent will never proceed into unknown or unplanned terrain. The local plans will fail immediately if this goal point is an obstacle, and in bounded time if other obstacles render the goal unreachable, in either case causing a new global plan to be computed over the updated information.

The PHR uses modified Planner structures, and alters the Replanner to have both a multi-threaded structure and the potential for more than two levels to the hierarchy. Like the HR, each Planner must be optimal with respect to the information over which it plans. Additionally, each Planner must finish execution promptly when a cancel command is received. In the case of incremental planning algorithms, state should be saved at this point so that a subsequent plan can continue with this stored information (possibly with a changed starting point or cost information). All levels are computed in separate threads, and their plans are asynchronously received by the thread responsible for the next lower level. Using the concept of the 'commit distance'[1] described in [5], all levels plan from the end of the committed path to their respective goal points.

Other than these changes, the system is identical in structure to the original HR, yet the modifications have important repercussions. First, the system is optimal with respect to the information in the state space of the highest level planner for which (2) applies. Secondly, the system is suited to the unconstrained multi-agent data fusion scenario, since the asynchronous higher level plans can incorporate the information from other agents prior to reaching these areas. By comparison, the HR only incorporates new information

[1]The length along a plan to which the agent's autonomous controllers are committed, subject to manual override or an independent safety system. Typically this is the section of a plan which has been communicated to the lower level controllers, and is thus no longer modifiable.

if the local plan fails, by then computing a new global plan. In the worst case scenario when all but the lowest level planner (which by requirement four must be guaranteed to be sufficiently reactive) take infinite time to repair their initial plans, the operation of the PHR is equivalent to the original HR.

V. Experimental Validation

To validate the PHR design it was implemented as described above, and tested in a simulation environment as well as on the CORD AGVs. This simulation was used to compare four hierarchical planning systems; the HR, the Incremental HR (IHR), the PerceptOR-style Replanner (PSR), and the PHR. The HR used the A^* algorithm for both local and global planning, whereas the IHR used the D^* algorithm for its global planning, potentially making these global repairs much faster. The PSR performed both a local and global plan simultaneously whenever new information was obtained, but was merged into the PHR framework to use the commit distance concept in addition. The vehicle would only stop if it reached the end of this commit distance before the combined plan was completed.

The measurements compared in this experiment were the total time taken to execute a mission, t_m, and the total cost of the path traversed, c_m. Since an agent must remain stationary at least until the first global plan is completed, t_m was measured as the duration between the initial global plan being completed and the goal being attained. The cumulative cost of nodes on the committed paths (using the cost at the time each node was committed to) provides the measure of c_m. The metric function used to evaluate this experiment is thus $M = k_1 \cdot t_m + k_2 \cdot c_m$, and the conclusions drawn depend on the weights k_1 and k_2 in a particular scenario.

Three scenarios were considered; a single vehicle mission, this same mission with a simulated aerial vehicle travelling in a straight line from the starting point towards the goal, and this same mission but with the aerial vehicle travelling from the goal towards the starting point. In Fig. 3 the colours of results pertaining to each of these scenarios are blue, green, and red respectively. For the CORD program and the simulation environment, the vehicle's velocity is a function of the cost of each node over which it travels, however the aerial vehicle was configured to travel at a constant 4m/s. The primary vehicle's velocity was determined by a cubic function of the cost, limited between a minimum of 0.5m/s and a maximum of 2.0m/s.

The cost of unknown regions was set at 20% of the maximum traversable cost, as this provides an empirically sensible trade-off between exploratory behaviour and traversal of known safe terrain, and yields a velocity of 1.8m/s using the function described. Given the sensor configuration on the vehicle (and replicated in the simulator), it is not expected that the vehicle would ever traverse unknown regions, since the sensors perceive these areas prior to the local plan committing to travel through them. A cost (which may be zero) must still be assigned to unknown areas however, so that the global plans are not limited to previously perceived regions.

The simulator loads regions of sensory data logged in previous trials so that they correspond with what the real vehicle's sensors would have perceived given the current pose. This means that regions that are out of the sensors' field of view remain unperceived in the simulator, an effect which is seen as 'holes' in the overlaid CostMap data seen in the first video attached to this paper[2].

Each experimental scenario was repeated five times for each planner, as slight changes in planning computation time alter what the agent observes, which in turn alters the plans computed, leading to variation in the results even for the same simulated mission.

A. Simulation Results

The results of this first experiment are shown in Fig. 3, and overall show almost no statistically significant differences between the performance of each planning system, save for a few important exceptions. Firstly, the c_m values for the single vehicle scenario (blue) are significantly higher than the multi-vehicle scenarios (green and red). This illustrates the intuitive result that having multiple agents perceive more of the environment allows for lower cost paths to be computed and executed. Secondly, the PSR has a significantly higher t_m value than any other system in the second multi-vehicle scenario (red). This increase in computation time is more clear for this second scenario, as the most computationally expensive plans occur in the early stages of the mission, when the path lengths (and thus the total plan repair time for the D^* algorithm) are higher.

There are several reasons for the insignificant differences between each planning system in this simulation. Primarily, this is because the results are highly dependent on the data observed by the vehicle, and by extension on the geometry of the environment. Minor changes to the regions perceived by the vehicle significantly alter the path geometry, since many otherwise traversable regions remain at the unknown cost, and are thus never considered. In addition, the difference in t_m between each planning system is only a small proportion of the total, and in most cases is less than the variance in t_m caused by other uncontrolled variables in the simulation.

Finally, since the choice of environment and scenarios did not execute the worst-case performance for any of the systems, the differences between them were less apparent. Using a more contrived scenario would more clearly illustrate the differences between each system. For example, having the vehicle drive into a cul-de-sac, would see the PSR and PHR adapt their global paths once the cul-de-sac closure was perceived by the aerial vehicle, whereas the HR and IHR would not. Alternatively, having two aerial vehicles repeatedly re-observe the region near the goal with slightly differing cost values would cause the PSR to continually stop moving while repairing the global plan. In a combination of

[2]Available from `http://www-personal.acfr.usyd.edu.au/t.allen/ICRA2009/PHR%20Simulation.mpg/avi`

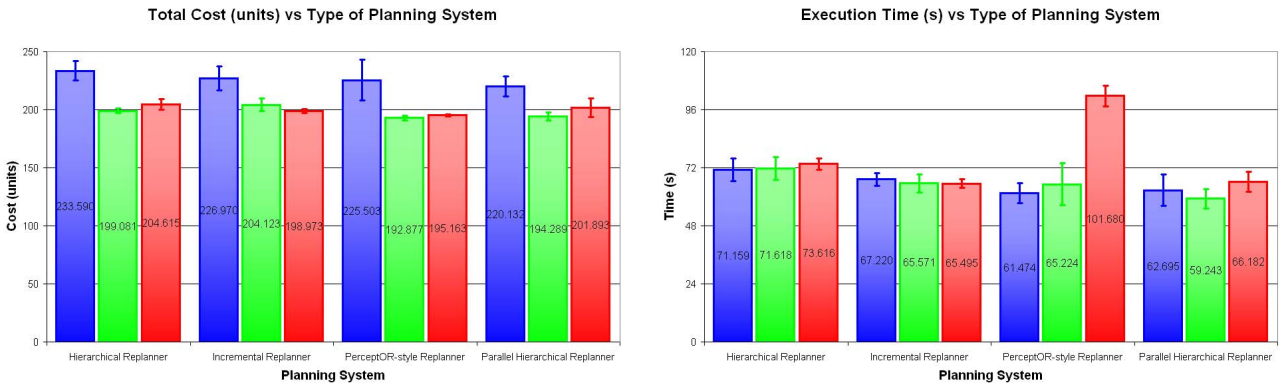


Fig. 3. Results of comparative experiment between four hierarchical planning systems. The left graph shows the total cost of the mission, c_m, and the right shows the execution time, t_m, with the error bars showing the 95% confidence interval. Blue represents the single vehicle scenario, green the simulated aerial vehicle travelling towards the goal, and red the aerial vehicle travelling towards the starting position of the ground vehicle.

such scenarios, the PHR could reasonably be expected to exhibit far lower t_m values than the other three approaches, and lower c_m values than all but the PSR.

It is concluded that in scenarios where following the initial global plan will result in high t_m or c_m values, both the PHR and PSR are better choices than the HR or IHR. However, if the scenario is also such that the global plans cannot be guaranteed to be sufficiently reactive, then the PHR is a better choice than a PSR for any metric where t_m has a higher weight than c_m.

B. Field Demonstration Results

The PHR was also implemented on the CORD AGVs, in conjunction with a new framework for the management of sensor information and uncertainties between multiple agents. The implementation used a two-level hierarchy, using the A^* and D^* algorithms for local and global levels respectively. This demonstration used the PHR to successfully navigate the CORD AGVs, simultaneously fusing the data obtained from both vehicles. The second video attached to this paper[3] shows video footage of two vehicles adapting to a dynamic obstacle which blocks the lead vehicle's original plan. The chase vehicle is then seen to adapt to an alternative route earlier than would be possible without data fusion.

VI. Future Work

The main area of future work is to replicate the simulation experiments in the field. The field testing area comprises several square kilometres of unstructured outdoor terrain, in which natural cul-de-sacs are often found. This should allow the types of scenarios described above to be performed, helping to more effectively distinguish the four planning systems. Additional work will assess the scalability of the results to systems with greater numbers of agents.

[3]Available from `http://www-personal.acfr.usyd.edu.au/t.allen/ICRA2009/PHR%20Field%20Demonstration.wmv/mp4`

VII. Conclusion

This paper has described the the use of asynchronous communication between the levels in a hierarchical planning system, known as the PHR, and validated the approach by simulation and on the CORD AGVs. The approach was seen to be well suited to the multi-agent data fusion scenario, particularly for unconstrained configurations of multiple agents that cause alternative approaches to increase in computation time.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006, available from http://planning.cs.uiuc.edu/.

[2] R. Jensen and M. Veloso, "Interleaving deliberative and reactive planning in dynamic multi-agent domains," in *Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures*. AAAI Press, Oct. 1998.

[3] K. C. Kluge and M. K. Morgenthaler, "Multi-horizon reactive and deliberative path planning for autonomous cross-country navigation," G. R. Gerhart, C. M. Shoemaker, and D. W. Gage, Eds., vol. 5422, no. 1. SPIE, 2004, pp. 461–472.

[4] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d* algorithm," *Journal of Field Robotics*, vol. 23, no. 2, pp. 79–101, 2 2006.

[5] T. Allen, J. Underwood, and S. Scheding, "A path planning system for autonomous ground vehicles operating in unstructured dynamic environments," in *Australasian Conference on Robotics and Automation*, 2007.

[6] P. Tompkins, A. Stentz, and D. Wettergreen, "Global path planning for mars rover exploration," in *IEEE Aerospace Conference*, 2004.

[7] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, "Global path planning on board the mars exploration rovers," in *IEEE Aerospace Conference*, 2007.

[8] A. Kelly, A. Stentz, O. Amidi, M. W. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. M. Vallidis, and R. Warner, "Toward reliable off road autonomous vehicles operating in challenging environments," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 449–483, May 2006.

[9] R. A. Brooks, "A robust layered control system for a mobile robot," Cambridge, MA, USA, Tech. Rep., 1985.

[10] J. Blythe and W. Scott-Reilly, "Integrating reactive and deliberative planning for agents," Pittsburgh, PA, USA, Tech. Rep. CMU-CS-93-155, 1993. [Online]. Available: citeseer.ist.psu.edu/blythe93integrating.html

[11] R. A. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2006, pp. 3375 – 3380.

Appendix C

Tabulated Results of the Monte-Carlo Analysis

This appendix tabulates the results of the Monte-Carlo analysis of TOPE estimators described in Section 5.4. This appendix is divided into four sections; Section C.1 tabulates the results of the comparison experiments performed upon the real world cost data visualised in Figure 2.1, and Sections C.2 to C.4 tabulate the results for randomised simulated data using a fractal terrain generator with roughness parameters of $H = 0.2$, $H = 0.5$, and $H = 0.8$, respectively.

Within each section there are eight tables. The first two show the percentage of runs in which each TOPE estimator outperforms each baseline technique, and the percentage of runs in which each baseline technique outperforms each TOPE estimator. The next set of three tables shows the percentage of runs in which each TOPE estimator outperforms each baseline technique by margins of 5%, 10%, and 25%, respectively. The final set of three tables shows the percentage of runs in which each baseline technique outperforms each TOPE estimator by margins of 5%, 10%, and 25%, respectively.

C.1 Real World Data

Table C.1: Percentage of runs in which each comparison method beat each baseline for a Monte-Carlo simulation over real-world cost map data.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	60.6	21.8	61.8	41.8	13.4	36.7	65.1	66.6	46.6	45.4	41.8	70.7
B2	55.5	20.9	61.8	42.1	14.0	34.6	64.8	66.6	43.6	43.3	40.9	67.8
B3	21.8	16.7	55.2	39.4	11.6	23.0	56.7	62.4	35.8	35.2	35.8	59.4
B4	41.8	22.4	59.1	43.0	13.4	31.3	58.5	63.9	39.7	41.5	40.3	63.3
B1b	76.4	44.8	71.3	56.1	23.9	54.6	72.8	77.9	71.3	70.4	56.4	83.3
B2b	63.6	44.8	64.5	51.6	14.6	49.9	61.8	74.6	60.0	57.6	49.9	71.9
B3b	60.6	44.8	68.4	54.3	23.9	51.3	71.0	77.0	68.4	69.0	55.5	82.1
B4b	66.9	46.3	72.2	57.9	26.3	54.0	73.1	76.7	71.9	71.9	55.5	84.2
Min	11.6	13.4	45.4	31.9	4.8	15.2	45.1	55.5	25.7	23.3	29.9	45.1

Table C.2: Percentage of runs in which each baseline beat each comparison method for a Monte-Carlo simulation over real-world cost map data.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	23.0	28.1	63.0	41.2	8.4	27.2	24.2	17.6
H2	63.0	63.9	69.3	62.1	32.5	44.5	32.5	30.7
H3	23.9	23.9	31.6	26.3	12.8	26.3	15.8	11.3
H4	41.8	41.5	45.7	40.6	21.2	33.1	23.0	19.1
H1b	80.0	79.4	81.5	79.7	57.6	61.2	57.6	54.6
H2b	49.6	51.6	64.5	54.6	27.2	39.4	30.4	27.2
H3b	20.6	20.9	30.1	26.9	11.3	29.0	13.1	10.7
H4b	20.9	20.9	26.6	23.6	3.9	12.2	4.8	4.5
S1	44.8	47.8	57.0	51.6	19.7	36.1	22.7	18.5
S2	45.4	47.5	56.7	49.3	20.3	37.0	21.8	18.2
S3	49.9	50.7	56.1	51.6	26.9	36.7	27.8	27.2
S4	27.5	30.4	38.8	34.9	14.9	28.1	16.1	13.7
Min	2.4	2.7	8.4	6.0	1.2	3.9	2.4	2.7

Table C.3: Percentage of runs in which each comparison method beat each baseline by a margin of 5% for a Monte-Carlo simulation over real-world cost map data.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	3.6	11.0	38.2	28.7	13.4	16.1	40.3	47.8	29.6	29.6	31.0	48.4
B2	2.7	10.7	36.4	25.4	13.4	15.5	38.8	46.6	28.1	28.7	28.7	48.7
B3	2.1	10.4	27.2	21.2	11.6	13.1	30.4	40.3	23.3	23.0	23.6	40.3
B4	11.3	10.1	33.1	28.7	13.4	18.2	36.7	48.1	28.1	27.8	29.3	46.0
B1b	56.4	41.8	64.8	50.7	20.9	45.7	65.4	70.7	64.8	64.5	53.7	77.0
B2b	53.4	41.8	49.9	40.0	9.3	41.2	49.9	50.4	56.1	56.4	46.0	64.2
B3b	56.1	41.8	62.7	48.7	20.9	43.9	61.8	65.1	63.6	63.3	52.8	74.3
B4b	61.2	41.5	63.6	49.0	21.5	48.7	64.8	72.5	66.9	65.7	54.0	79.4
Min	1.8	6.6	12.5	11.9	4.5	6.0	17.3	23.3	15.2	14.9	18.2	28.1

Table C.4: Percentage of runs in which each comparison method beat each baseline by a margin of 10% for a Monte-Carlo simulation over real-world cost map data.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	2.7	9.3	18.5	14.0	13.4	11.0	19.7	30.4	18.2	22.1	19.4	36.4
B2	2.1	9.3	19.1	12.5	13.4	11.3	20.0	29.9	18.2	21.5	17.6	35.8
B3	2.1	9.0	13.7	9.9	11.6	10.4	15.8	20.6	17.6	18.5	14.6	28.4
B4	4.8	7.5	15.5	13.7	13.4	11.3	18.5	24.8	20.3	20.6	20.0	33.7
B1b	53.4	40.3	54.6	44.5	20.9	39.7	53.7	58.2	58.8	59.1	49.0	68.4
B2b	51.9	40.3	48.4	38.2	9.3	35.2	48.4	47.5	55.2	54.6	44.2	62.1
B3b	53.4	40.6	52.2	42.1	20.9	39.7	52.5	53.1	58.2	58.5	47.8	67.8
B4b	54.6	41.5	54.0	43.6	21.5	40.9	55.8	60.9	60.9	60.3	49.3	71.3
Min	1.8	5.7	8.4	6.6	4.5	6.0	9.9	11.3	12.5	11.6	13.4	22.4

Table C.5: Percentage of runs in which each comparison method beat each baseline by a margin of 25% for a Monte-Carlo simulation over real-world cost map data.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	2.1	4.8	6.3	3.0	11.9	4.8	6.9	7.5	10.4	10.7	11.6	18.2
B2	2.1	4.8	6.3	3.0	11.9	4.8	6.6	7.5	10.1	9.6	11.6	18.2
B3	1.8	4.5	6.0	3.0	10.1	4.5	6.6	7.5	10.1	10.4	10.4	16.7
B4	2.1	4.8	6.3	3.6	11.9	4.8	6.9	9.0	11.3	10.7	12.2	18.5
B1b	43.0	30.1	42.7	34.6	19.4	32.8	44.2	42.7	49.3	48.4	40.3	58.8
B2b	41.5	31.6	43.0	32.8	9.3	29.9	44.5	41.5	47.5	46.0	37.9	55.2
B3b	43.0	30.1	42.7	34.6	19.4	32.8	44.2	42.7	49.3	48.4	40.3	58.8
B4b	43.9	30.7	42.4	35.2	20.0	32.5	43.6	43.6	49.9	47.5	40.0	59.1
Min	1.8	3.0	3.6	2.1	4.5	1.5	3.0	2.7	7.2	6.3	10.1	13.7

Table C.6: Percentage of runs in which each baseline beat each comparison method by a margin of 5% for a Monte-Carlo simulation over real-world cost map data.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.9	0.9	4.2	6.6	3.6	21.2	3.6	4.5
H2	47.5	51.0	54.6	48.7	25.7	40.0	28.1	26.6
H3	12.2	12.8	15.2	13.4	7.2	16.4	8.7	7.8
H4	34.0	34.6	37.6	29.6	17.0	24.8	20.0	14.6
H1b	80.0	79.4	81.5	79.7	50.1	49.9	50.1	45.7
H2b	34.3	34.3	40.0	36.4	14.0	24.2	15.5	18.2
H3b	9.6	9.6	12.2	12.2	4.8	16.4	6.9	5.7
H4b	12.2	12.2	14.6	11.9	0.9	7.2	0.9	1.8
S1	23.3	24.5	32.5	26.6	11.9	27.5	14.6	7.8
S2	24.5	25.7	32.8	26.3	12.2	23.9	15.2	11.9
S3	35.8	36.4	42.1	34.3	20.9	28.4	22.7	20.0
S4	12.2	13.1	17.0	10.7	6.6	12.5	10.4	4.8
Min	0.0	0.0	1.5	0.0	0.0	0.0	0.0	0.0

Table C.7: Percentage of runs in which each baseline beat each comparison method by a margin of 10% for a Monte-Carlo simulation over real-world cost map data.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.9	0.9	3.3	4.2	3.6	13.7	3.6	3.3
H2	44.2	43.9	48.7	43.0	22.7	37.6	23.6	20.6
H3	9.9	10.4	12.8	10.7	6.0	15.2	7.5	4.2
H4	27.8	27.8	29.3	27.2	12.8	20.0	13.7	14.0
H1b	78.2	78.5	81.5	79.7	40.6	42.1	43.3	39.1
H2b	31.3	32.2	34.3	31.9	9.9	20.0	9.9	7.5
H3b	7.2	8.4	10.1	9.0	4.2	13.7	5.7	3.9
H4b	9.3	9.3	10.7	9.3	0.9	6.0	0.9	0.3
S1	11.6	13.1	16.7	12.5	6.9	17.9	9.3	4.5
S2	10.4	11.0	15.5	11.6	6.6	13.7	9.0	6.6
S3	30.7	30.7	33.1	29.0	19.1	24.2	20.0	16.7
S4	5.4	5.7	7.2	3.9	3.3	7.2	4.5	2.1
Min	0.0	0.0	1.5	0.0	0.0	0.0	0.0	0.0

Table C.8: Percentage of runs in which each baseline beat each comparison method by a margin of 25% for a Monte-Carlo simulation over real-world cost map data.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.6	0.6	2.1	0.6	3.6	9.6	3.6	3.3
H2	30.1	30.1	34.0	29.6	14.0	31.0	16.4	15.5
H3	7.5	7.5	8.7	7.2	4.2	11.9	4.2	3.6
H4	21.2	21.5	23.6	23.6	11.3	18.8	11.3	11.0
H1b	65.7	65.7	69.6	67.8	20.3	34.9	21.2	19.7
H2b	19.7	19.4	22.1	20.3	3.9	12.8	3.9	1.8
H3b	6.6	6.6	7.8	6.0	4.2	10.7	4.2	3.9
H4b	7.8	7.8	9.6	8.1	0.9	6.0	0.9	0.3
S1	2.7	2.7	4.2	3.3	0.9	6.6	1.2	0.6
S2	4.5	4.5	6.0	5.1	3.3	9.0	4.8	2.4
S3	25.4	25.7	27.5	26.0	17.0	20.3	17.0	16.4
S4	1.2	1.5	3.0	1.2	1.2	2.7	1.2	0.6
Min	0.0	0.0	1.5	0.0	0.0	0.0	0.0	0.0

C.2 Randomised Fractal Terrain with Low Roughness

Table C.9: Percentage of runs in which each comparison method beat each baseline for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	55.8	26.9	18.2	31.8	65.7	33.5	21.1	38.0	68.6	49.2	68.6	63.2
B2	54.5	20.2	15.7	31.8	61.2	30.6	17.8	36.4	67.8	45.0	66.1	59.5
B3	48.3	10.7	10.7	21.1	51.2	25.2	15.7	26.0	55.8	36.4	57.4	52.5
B4	43.8	11.6	12.8	28.5	58.7	26.4	16.5	31.8	60.3	37.6	59.9	54.1
B1b	41.7	33.1	20.7	28.5	46.3	28.1	22.3	34.3	53.7	36.8	60.7	57.9
B2b	46.3	33.5	30.2	36.0	23.6	37.6	33.1	40.1	50.0	47.5	67.4	62.8
B3b	39.3	34.7	22.3	28.5	39.7	28.5	24.0	33.5	52.5	34.7	59.5	54.5
B4b	39.3	35.1	29.8	35.5	51.7	29.3	33.1	36.4	52.5	52.5	74.8	67.8
Min	31.4	5.0	8.3	12.8	10.3	15.7	12.4	15.7	40.5	17.4	35.1	31.4

Table C.10: Percentage of runs in which each baseline beat each comparison method for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	2.1	3.3	9.1	13.2	25.6	42.6	29.8	29.8
H2	25.6	32.2	43.0	40.5	32.6	53.7	32.2	30.6
H3	32.6	35.1	41.3	38.4	44.2	57.0	45.0	33.1
H4	24.0	24.4	35.5	26.4	37.6	51.7	39.7	31.4
H1b	26.9	31.4	41.3	33.5	47.5	38.4	53.7	42.6
H2b	25.2	28.5	35.1	32.2	38.4	50.8	40.1	34.3
H3b	31.4	34.7	38.0	36.0	44.2	54.5	44.6	31.0
H4b	18.2	20.2	31.8	24.8	33.1	48.3	34.7	30.2
S1	7.9	9.1	21.5	16.9	25.2	44.6	26.9	26.4
S2	27.3	31.4	41.7	39.3	44.2	44.6	47.1	26.9
S3	22.7	25.2	34.3	31.8	32.6	31.4	34.3	18.6
S4	22.7	26.4	34.7	31.8	31.0	32.6	34.7	19.8
Min	0.0	0.0	0.0	0.4	4.5	8.7	4.5	4.5

Table C.11: Percentage of runs in which each comparison method beat each baseline by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	17.4	8.7	6.6	12.0	44.6	10.3	6.6	12.4	43.4	28.9	44.6	39.7
B2	19.0	7.0	5.4	10.7	43.4	10.3	5.8	12.0	44.6	28.1	43.4	38.0
B3	9.5	5.4	2.1	9.1	41.7	9.5	2.9	9.1	38.8	27.7	40.9	36.8
B4	23.1	5.0	2.5	12.0	43.0	9.9	4.1	10.3	43.0	28.1	43.0	37.2
B1b	29.3	22.3	9.1	19.4	30.6	4.1	8.7	21.1	43.4	19.4	40.1	36.0
B2b	32.6	25.2	12.8	22.7	11.6	7.0	10.7	24.8	40.5	17.4	38.8	34.7
B3b	28.5	21.5	8.3	17.8	27.3	2.5	7.4	19.4	42.1	16.5	36.8	32.2
B4b	31.8	24.4	11.6	21.9	32.2	5.8	9.9	24.0	42.1	24.8	49.2	42.6
Min	7.9	1.7	0.4	4.5	7.0	0.8	0.8	1.7	23.1	5.0	14.0	12.4

Table C.12: Percentage of runs in which each comparison method beat each baseline by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	12.4	8.3	3.7	10.7	44.2	10.3	4.1	9.9	36.8	28.1	43.0	37.6
B2	11.6	5.8	2.5	9.1	43.0	9.5	2.9	7.4	37.6	26.9	41.7	36.4
B3	9.1	4.5	1.7	7.9	41.7	9.5	2.9	7.0	31.4	27.3	40.9	36.4
B4	16.5	3.7	2.1	9.1	43.0	9.5	3.3	9.5	40.1	28.1	43.0	37.2
B1b	25.6	18.6	5.0	15.7	30.6	3.7	6.2	17.4	40.5	18.2	33.1	30.6
B2b	28.5	20.2	7.9	19.0	11.6	5.4	7.4	22.3	38.8	12.0	23.6	20.2
B3b	24.0	17.8	4.1	14.0	27.3	2.1	5.0	14.5	39.3	15.3	29.8	27.3
B4b	27.7	19.8	7.0	16.9	32.2	4.1	6.2	19.8	40.1	20.2	37.2	34.3
Min	4.5	1.2	0.4	3.7	7.0	0.8	0.4	1.2	15.7	3.3	9.5	6.6

Table C.13: Percentage of runs in which each comparison method beat each baseline by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	12.0	4.1	2.5	10.3	44.2	10.3	4.1	8.3	28.9	28.1	43.0	37.6
B2	10.3	2.9	1.2	8.7	43.0	9.5	2.9	7.0	28.1	26.9	41.7	36.4
B3	8.7	2.9	1.2	7.9	41.7	9.5	2.9	7.0	26.9	27.3	40.9	36.4
B4	9.9	2.9	2.1	7.9	43.0	9.5	3.3	7.4	29.3	27.7	42.6	36.8
B1b	12.4	2.5	1.7	7.0	30.6	3.3	3.3	5.0	25.2	17.8	30.2	25.6
B2b	19.8	11.2	2.5	12.4	11.6	2.9	3.3	11.2	27.7	6.6	16.5	14.0
B3b	10.3	0.0	0.4	5.4	27.3	1.7	1.7	2.1	21.9	14.9	26.9	22.3
B4b	19.0	10.3	1.7	10.7	32.2	1.7	3.7	9.1	30.2	17.4	33.5	28.5
Min	4.1	0.0	0.0	2.5	7.0	0.8	0.4	0.4	6.6	3.3	8.7	6.6

Table C.14: Percentage of runs in which each baseline beat each comparison method by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	1.7	1.7	1.2	4.5	11.2	32.6	12.8	12.8
H2	14.0	12.8	17.4	7.4	14.0	35.5	15.3	14.0
H3	21.1	20.7	24.4	19.8	26.4	43.8	29.8	20.2
H4	17.4	16.1	19.4	15.7	19.8	35.5	21.9	14.9
H1b	24.4	24.4	26.4	24.8	21.9	7.0	24.4	5.4
H2b	22.3	23.1	24.8	21.5	22.3	36.4	24.8	11.6
H3b	20.7	21.1	23.1	19.8	26.4	42.1	28.9	19.0
H4b	13.2	13.6	17.8	13.6	16.5	35.5	17.8	13.6
S1	1.7	2.1	2.9	3.7	4.1	19.8	4.5	4.1
S2	21.5	21.5	24.0	21.1	18.2	16.9	20.7	4.1
S3	17.4	19.0	22.3	20.2	9.5	7.9	11.2	2.5
S4	17.8	19.4	22.3	19.0	9.5	9.9	10.7	2.1
Min	0.0	0.0	0.0	0.0	0.4	1.7	0.4	0.4

Table C.15: Percentage of runs in which each baseline beat each comparison method by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	1.7	1.7	1.2	2.9	11.2	32.6	12.8	12.8
H2	10.3	7.0	14.9	3.3	14.0	35.5	15.3	14.0
H3	17.8	18.2	20.7	16.9	25.2	43.8	27.7	19.4
H4	12.0	12.4	15.7	11.6	17.4	34.3	19.4	14.0
H1b	21.1	22.7	24.8	22.7	19.0	7.0	19.4	4.1
H2b	20.2	21.1	22.3	19.4	21.1	36.4	23.1	11.6
H3b	18.2	18.6	19.8	18.2	24.4	42.1	26.4	18.6
H4b	10.3	9.9	13.2	9.5	15.3	34.3	16.1	12.4
S1	1.7	2.1	2.5	2.9	3.7	19.4	4.1	3.7
S2	16.9	17.4	21.5	19.0	11.6	16.9	12.4	4.1
S3	14.0	14.5	18.2	15.7	7.9	7.9	8.7	2.5
S4	14.0	12.8	16.9	15.3	7.9	9.9	8.7	2.1
Min	0.0	0.0	0.0	0.0	0.4	1.7	0.4	0.4

Table C.16: Percentage of runs in which each baseline beat each comparison method by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.2$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	1.7	1.7	1.2	0.8	11.2	32.6	12.8	12.8
H2	0.8	0.8	2.1	0.4	14.0	35.5	15.3	14.0
H3	9.9	9.5	12.4	10.7	19.4	41.7	21.9	17.4
H4	7.0	7.4	7.9	7.0	12.4	33.9	14.5	13.2
H1b	15.3	15.7	17.4	18.6	11.2	7.0	12.0	4.1
H2b	15.3	16.9	17.8	15.7	16.1	36.4	18.2	11.6
H3b	14.0	14.0	15.7	13.2	21.9	42.1	24.0	18.6
H4b	4.1	5.0	7.0	4.5	13.2	34.3	14.0	12.4
S1	1.2	1.7	2.1	2.1	3.7	19.4	4.1	3.7
S2	12.8	13.6	15.7	15.3	8.7	16.9	9.5	4.1
S3	5.8	6.6	10.3	6.6	4.1	7.9	4.5	2.5
S4	6.6	7.4	9.9	4.5	3.7	9.9	4.1	2.1
Min	0.0	0.0	0.0	0.0	0.4	1.7	0.4	0.4

C.3 Randomised Fractal Terrain with Medium Roughness

Table C.17: Percentage of runs in which each comparison method beat each baseline for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	52.5	13.1	13.9	23.4	72.5	27.5	13.9	27.9	52.9	24.6	61.9	41.4
B2	52.0	5.7	9.4	18.9	65.2	24.2	10.2	24.2	48.4	20.9	51.6	33.2
B3	35.2	5.3	5.7	12.3	53.7	18.0	9.0	13.5	35.2	17.2	43.4	27.5
B4	35.7	7.4	6.1	16.4	61.5	22.5	8.2	18.9	47.5	21.3	52.5	32.8
B1b	48.4	25.4	18.0	27.9	59.4	22.5	17.2	29.1	57.0	27.5	61.5	42.6
B2b	46.7	29.9	25.4	33.6	33.2	37.3	25.4	35.2	58.2	35.7	77.9	57.8
B3b	48.4	25.4	17.6	27.9	57.8	23.0	17.2	29.1	56.6	27.9	60.7	42.6
B4b	50.4	27.9	21.7	32.0	61.9	31.1	24.2	33.6	58.6	34.4	72.1	54.5
Min	27.9	0.0	2.9	9.8	24.6	13.5	6.6	8.6	29.1	10.2	38.9	23.8

Table C.18: Percentage of runs in which each baseline beat each comparison method for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.4	0.8	17.6	17.2	6.6	37.7	6.6	7.0
H2	31.6	38.9	44.3	37.3	23.4	48.4	23.0	23.0
H3	24.2	34.8	43.9	36.1	30.7	52.9	30.7	24.6
H4	21.7	29.9	41.0	30.3	21.7	44.7	21.7	17.6
H1b	23.8	31.1	42.6	34.8	34.8	27.5	36.5	30.3
H2b	17.2	26.2	36.9	27.9	28.3	41.0	27.9	18.0
H3b	24.2	34.0	40.6	34.0	31.6	53.3	31.6	22.1
H4b	18.9	24.6	38.5	29.9	21.3	44.7	21.3	17.2
S1	18.0	22.5	35.7	23.4	13.9	27.5	14.3	12.3
S2	26.2	35.7	43.9	35.2	29.5	43.9	29.1	18.4
S3	23.4	33.6	41.8	32.8	23.8	13.1	24.6	13.1
S4	24.2	32.4	42.2	32.8	23.0	23.4	23.0	11.1
Min	0.0	0.4	4.9	5.3	3.7	2.0	3.7	2.0

Table C.19: Percentage of runs in which each comparison method beat each baseline by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	18.0	9.4	3.7	10.2	60.2	8.6	3.3	12.3	37.7	14.8	49.6	29.9
B2	14.8	3.3	2.0	7.0	52.9	7.0	2.0	7.4	34.4	13.1	42.2	22.1
B3	5.3	2.9	1.2	5.3	48.0	6.6	1.6	3.7	26.2	12.7	36.9	21.3
B4	23.0	5.3	1.6	7.0	55.3	9.4	2.0	10.7	35.2	16.0	44.7	25.4
B1b	31.1	16.8	8.6	19.7	48.0	7.0	6.6	20.1	45.1	14.3	44.3	25.0
B2b	34.8	22.1	14.8	25.4	20.9	9.4	14.8	27.5	42.6	14.3	43.4	28.3
B3b	31.1	16.8	8.2	19.7	48.0	7.0	6.6	19.7	44.7	14.3	44.3	25.0
B4b	33.2	20.5	10.7	25.4	50.4	8.2	12.3	26.2	44.3	14.3	52.9	34.0
Min	3.3	0.0	0.0	4.1	16.8	0.4	0.8	2.5	14.3	1.2	16.0	4.1

Table C.20: Percentage of runs in which each comparison method beat each baseline by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	17.6	9.4	2.5	9.4	60.2	8.6	2.0	11.1	37.3	14.8	49.6	29.9
B2	9.8	1.2	0.8	5.7	52.9	7.0	0.8	5.7	30.7	13.1	41.8	22.1
B3	4.5	1.2	1.2	4.9	48.0	6.6	1.2	3.7	24.6	12.7	36.9	21.3
B4	13.9	4.1	1.2	6.1	55.3	9.4	1.6	8.2	32.0	16.0	44.7	25.0
B1b	25.0	10.7	6.1	12.7	48.0	4.5	4.5	13.9	40.2	10.2	41.0	21.3
B2b	30.7	20.1	8.6	20.1	20.9	4.1	6.6	21.7	36.9	5.7	26.6	15.6
B3b	25.0	10.2	5.7	11.9	48.0	4.5	4.5	13.9	39.8	10.2	41.0	21.3
B4b	28.3	18.4	6.6	17.2	50.4	5.3	4.9	16.8	40.2	9.4	45.9	25.8
Min	2.0	0.0	0.0	1.6	16.8	0.0	0.4	0.4	13.5	1.2	10.7	2.9

Table C.21: Percentage of runs in which each comparison method beat each baseline by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	17.2	8.6	2.0	9.0	60.2	8.6	2.0	10.7	35.2	14.8	49.2	29.5
B2	9.4	1.2	0.8	5.3	52.9	7.0	0.8	5.3	27.5	13.1	41.8	22.1
B3	4.5	1.2	1.2	4.9	48.0	6.6	1.2	3.7	22.5	12.7	36.9	21.3
B4	11.9	3.7	1.2	5.7	55.3	9.4	1.2	7.8	30.3	15.6	44.3	24.6
B1b	12.7	0.4	0.4	1.6	48.0	2.5	0.8	2.0	26.6	8.6	37.3	17.2
B2b	23.8	11.9	1.2	4.9	20.9	0.0	2.0	8.2	27.0	1.2	13.9	3.7
B3b	12.7	0.0	0.0	1.6	48.0	2.5	0.8	2.0	26.6	8.6	37.3	17.2
B4b	21.7	12.7	1.6	6.1	50.4	3.3	2.0	9.4	35.2	7.0	39.8	20.1
Min	2.0	0.0	0.0	0.4	16.8	0.0	0.4	0.4	7.4	1.2	8.6	2.9

Table C.22: Percentage of runs in which each baseline beat each comparison method by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.0	0.0	0.0	0.0	3.7	31.6	4.1	4.5
H2	13.5	17.2	26.6	13.9	7.0	34.4	6.6	9.0
H3	10.7	18.9	25.8	15.6	20.5	41.0	20.1	11.5
H4	11.9	17.2	23.4	14.3	11.5	34.4	11.5	7.4
H1b	18.4	25.8	30.7	23.4	20.5	4.9	20.5	9.8
H2b	13.9	21.7	27.0	19.7	19.7	34.8	19.7	7.8
H3b	13.1	20.9	26.2	18.9	23.8	42.2	23.8	13.1
H4b	9.8	13.5	18.0	12.3	11.1	34.4	11.1	7.0
S1	2.5	3.3	6.6	2.9	2.5	15.2	2.9	1.6
S2	10.2	18.4	27.5	18.4	14.8	29.1	14.8	6.1
S3	9.4	18.0	26.2	16.8	14.8	6.1	14.8	1.6
S4	9.8	18.0	25.8	16.8	14.8	16.0	14.8	2.5
Min	0.0	0.0	0.0	0.0	0.0	1.6	0.0	0.0

Table C.23: Percentage of runs in which each baseline beat each comparison method by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.0	0.0	0.0	0.0	2.0	31.6	2.0	4.5
H2	7.4	4.1	24.2	7.8	7.0	34.4	6.6	9.0
H3	10.2	17.6	23.8	14.8	17.2	41.0	16.8	9.0
H4	9.4	12.7	19.3	9.8	10.2	34.4	10.2	7.4
H1b	13.9	21.7	27.9	21.3	18.4	4.9	18.4	5.3
H2b	12.7	20.1	25.8	18.9	17.6	34.8	17.6	7.8
H3b	12.3	20.5	26.2	18.0	20.1	42.2	20.1	11.5
H4b	6.1	9.4	14.3	7.4	8.2	34.4	8.2	5.7
S1	1.2	1.2	1.2	1.2	1.2	15.2	1.2	1.2
S2	8.2	16.4	21.3	14.3	14.8	29.1	14.8	2.5
S3	7.8	15.2	20.9	12.3	13.5	6.1	13.5	0.4
S4	8.2	15.6	20.9	12.7	14.3	16.0	14.3	0.8
Min	0.0	0.0	0.0	0.0	0.0	1.6	0.0	0.0

Table C.24: Percentage of runs in which each baseline beat each comparison method by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.5$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	0.0	0.0	0.0	0.0	2.0	31.6	2.0	4.5
H2	0.8	0.8	5.7	0.8	4.9	34.4	4.5	7.0
H3	7.0	13.1	21.7	12.7	13.1	41.0	12.7	9.0
H4	7.0	9.4	15.2	6.6	6.1	34.4	6.1	6.1
H1b	11.5	17.6	23.0	16.0	14.3	2.9	14.3	2.0
H2b	10.2	16.8	23.4	17.2	11.5	34.8	11.5	7.8
H3b	9.8	16.8	25.4	14.8	13.9	42.2	13.9	11.5
H4b	5.3	7.4	10.7	6.6	4.9	34.4	4.9	5.3
S1	0.4	0.4	0.4	0.4	0.4	15.2	0.4	0.4
S2	7.8	13.9	18.4	12.3	11.1	29.1	11.1	2.5
S3	6.6	12.3	16.8	12.3	3.3	6.1	3.7	0.4
S4	6.6	12.3	17.2	11.5	3.3	16.0	3.7	0.4
Min	0.0	0.0	0.0	0.0	0.0	1.6	0.0	0.0

C.4 Randomised Fractal Terrain with High Roughness

Table C.25: Percentage of runs in which each comparison method beat each baseline for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	47.1	17.6	31.7	39.8	30.3	28.5	40.3	51.1	61.5	41.2	52.5	60.6
B2	43.4	14.5	30.8	39.8	30.3	29.0	39.8	49.8	60.2	37.6	53.4	60.2
B3	34.8	14.0	30.3	37.6	28.5	25.8	37.6	47.5	57.5	33.5	49.3	52.5
B4	36.7	12.2	30.3	36.2	31.2	25.3	37.6	46.2	58.4	39.4	52.0	54.3
B1b	53.8	38.9	46.2	53.4	43.9	43.9	55.7	57.0	72.9	61.1	67.9	71.0
B2b	52.5	37.1	50.2	53.4	19.5	52.0	57.9	57.0	71.9	64.7	71.9	74.7
B3b	52.5	38.9	45.7	53.4	40.7	43.9	55.7	57.0	72.4	60.6	67.9	71.0
B4b	52.5	38.9	48.4	53.4	46.6	47.5	55.7	57.0	72.4	65.6	71.9	74.2
Min	27.1	10.4	29.9	35.3	10.9	22.2	37.1	45.2	56.6	32.6	49.3	51.1

Table C.26: Percentage of runs in which each baseline beat each comparison method for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	14.9	19.5	29.0	26.2	4.5	27.1	7.2	5.4
H2	40.3	44.8	46.2	45.2	11.3	41.2	12.7	11.3
H3	28.1	30.3	31.2	30.8	10.0	30.3	11.3	7.2
H4	23.1	24.9	27.1	28.1	2.3	20.8	4.1	1.8
H1b	53.4	55.2	57.0	53.8	30.3	42.1	34.4	27.1
H2b	31.2	33.0	36.7	36.7	13.6	26.7	15.4	8.1
H3b	21.3	23.5	26.2	25.8	5.9	25.3	7.2	4.1
H4b	13.6	15.8	17.6	18.6	1.4	19.0	3.6	0.9
S1	13.6	14.9	17.2	15.8	2.7	18.1	3.6	2.7
S2	28.1	31.7	35.7	29.4	8.6	21.3	11.3	3.2
S3	29.9	30.3	34.4	29.4	5.4	10.9	6.3	1.4
S4	15.8	16.7	24.4	22.6	5.4	14.9	6.8	2.3
Min	0.9	1.4	3.6	5.4	0.9	6.8	1.8	0.9

Table C.27: Percentage of runs in which each comparison method beat each baseline by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	8.1	7.7	22.6	33.0	28.1	12.2	27.6	41.2	49.8	27.1	38.9	44.3
B2	7.7	8.6	21.3	32.6	27.6	13.6	26.2	39.8	47.5	24.4	37.1	43.0
B3	4.5	4.5	19.0	29.9	26.2	9.5	22.2	35.7	42.1	19.5	33.5	39.4
B4	14.0	7.7	18.6	28.5	29.0	12.7	24.9	36.7	48.0	26.2	36.2	41.2
B1b	49.3	32.1	37.1	51.1	34.4	38.5	49.3	54.3	69.7	53.8	61.1	67.0
B2b	49.3	32.6	43.9	50.7	10.9	47.1	53.4	53.4	68.3	55.2	67.4	72.4
B3b	49.3	31.7	36.7	51.1	33.5	38.5	48.9	54.8	69.7	52.9	60.6	67.9
B4b	49.3	32.6	42.5	51.1	36.7	43.0	51.1	54.8	69.7	60.6	68.8	72.9
Min	4.1	1.8	15.8	26.7	7.2	6.8	21.7	32.6	40.7	16.7	31.7	38.5

Table C.28: Percentage of runs in which each comparison method beat each baseline by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	6.3	2.7	11.8	25.8	28.1	7.2	19.9	32.6	36.2	20.4	32.1	33.0
B2	5.4	2.7	11.8	25.8	27.6	7.7	19.5	30.3	33.9	17.6	30.8	33.0
B3	4.5	1.4	8.6	19.5	26.2	6.3	16.7	22.6	29.4	15.8	29.4	29.0
B4	9.5	5.0	11.3	22.6	29.0	9.5	20.4	27.1	33.5	17.2	31.2	36.2
B1b	46.6	28.5	33.0	43.4	34.4	30.8	40.3	49.8	64.3	46.2	53.8	56.6
B2b	47.1	30.3	40.3	44.8	10.9	33.9	45.7	49.8	65.6	47.1	56.1	61.5
B3b	46.2	28.1	33.0	43.9	33.5	30.8	40.3	50.2	63.8	46.6	54.8	57.5
B4b	46.2	31.7	36.7	47.1	36.7	32.6	43.9	50.7	64.7	47.1	60.6	63.8
Min	4.1	0.9	7.7	18.1	7.2	5.9	15.8	19.5	26.2	10.4	25.8	25.3

Table C.29: Percentage of runs in which each comparison method beat each baseline by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	H1	**H2**	**H3**	**H4**	**H1b**	**H2b**	**H3b**	**H4b**	**S1**	**S2**	**S3**	**S4**
B1	6.3	2.3	4.5	7.2	28.1	4.1	5.9	9.0	20.4	13.6	26.7	20.8
B2	5.0	1.4	3.6	6.8	27.6	4.1	5.4	7.7	19.0	11.3	25.8	19.0
B3	4.5	0.9	2.7	5.4	26.2	3.2	4.5	5.9	16.3	10.0	24.4	17.6
B4	6.8	1.4	5.4	8.1	29.0	5.9	7.2	8.6	19.9	12.7	25.3	20.8
B1b	26.7	14.9	22.2	21.7	34.4	22.2	29.0	32.6	46.2	35.7	40.3	43.9
B2b	33.5	23.1	23.1	29.0	10.0	18.1	31.7	38.0	48.9	33.0	34.8	43.0
B3b	26.2	14.5	21.3	21.3	33.5	22.2	28.5	33.0	44.8	36.2	39.4	43.4
B4b	32.1	20.8	23.1	27.6	34.8	23.1	31.2	33.9	48.9	37.6	42.1	45.7
Min	4.1	0.0	0.9	2.7	7.2	0.5	2.3	5.4	12.2	4.5	13.1	11.3

Table C.30: Percentage of runs in which each baseline beat each comparison method by a margin of 5% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	8.1	9.5	10.4	11.8	3.6	24.9	5.0	3.2
H2	32.6	35.3	38.5	35.7	7.2	35.3	8.6	7.2
H3	26.7	29.0	30.3	29.0	9.5	29.4	10.4	6.8
H4	19.5	21.7	22.2	19.9	2.3	20.8	4.1	1.8
H1b	49.8	52.0	52.9	49.8	12.7	14.9	14.5	5.9
H2b	24.0	25.8	26.7	24.9	7.7	24.4	9.5	5.9
H3b	19.9	21.3	24.0	22.6	5.4	24.9	6.8	3.6
H4b	13.1	14.0	14.5	13.6	1.4	19.0	3.6	0.9
S1	6.8	7.2	8.6	8.1	1.8	16.3	2.3	1.4
S2	14.5	15.4	19.9	20.8	5.4	18.1	7.7	1.4
S3	20.8	22.2	23.1	22.6	5.4	10.4	5.9	0.9
S4	11.3	12.7	13.6	14.5	4.5	14.0	5.4	0.9
Min	0.0	0.0	0.0	0.5	0.9	4.5	1.8	0.9

Table C.31: Percentage of runs in which each baseline beat each comparison method by a margin of 10% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	8.1	9.0	10.0	11.3	3.6	24.9	5.0	3.2
H2	26.7	26.2	33.0	26.2	7.2	35.3	8.6	7.2
H3	21.7	24.9	26.2	23.5	9.5	29.4	10.4	6.8
H4	15.8	18.6	18.1	14.5	2.3	20.8	4.1	1.8
H1b	44.8	47.5	51.1	49.3	12.7	9.5	14.5	5.4
H2b	21.7	24.9	25.3	22.2	7.7	24.4	9.5	5.9
H3b	17.6	20.4	20.8	16.7	5.4	24.9	6.8	3.6
H4b	11.8	12.7	12.2	10.4	1.4	19.0	3.6	0.9
S1	4.1	4.1	4.5	4.1	1.4	15.8	1.8	0.9
S2	13.1	14.5	15.8	15.4	5.4	18.1	7.2	0.9
S3	18.6	20.4	20.4	18.1	4.5	10.4	5.0	0.9
S4	9.0	10.9	11.3	12.2	3.2	14.0	4.1	0.9
Min	0.0	0.0	0.0	0.0	0.9	4.1	1.8	0.5

Table C.32: Percentage of runs in which each baseline beat each comparison method by a margin of 25% for randomised fractal terrain data with a roughness parameter of $H = 0.8$.

	B1	**B2**	**B3**	**B4**	**B1b**	**B2b**	**B3b**	**B4b**
H1	8.1	9.0	10.0	9.0	3.6	24.9	5.0	3.2
H2	16.3	17.6	18.6	15.8	7.2	35.3	8.6	7.2
H3	14.9	16.3	16.7	16.3	5.0	29.4	5.9	4.5
H4	10.0	11.8	11.8	10.9	2.3	20.8	4.1	1.8
H1b	36.7	39.4	39.4	34.8	10.4	9.5	12.2	5.4
H2b	16.7	17.6	19.5	15.8	5.4	24.4	7.2	3.6
H3b	7.2	9.0	10.4	9.0	3.2	24.9	4.5	1.4
H4b	8.6	9.5	10.0	7.7	1.4	19.0	3.6	0.9
S1	1.8	1.8	1.4	0.0	1.4	15.8	1.8	0.9
S2	10.0	10.9	11.8	8.6	3.2	18.1	5.0	0.9
S3	14.9	15.4	17.6	13.6	3.2	10.4	3.6	0.9
S4	4.5	5.9	6.3	5.9	2.7	14.0	3.6	0.9
Min	0.0	0.0	0.0	0.0	0.9	4.1	1.8	0.5

Appendix D

Performance Matrices for the Monte-Carlo Analysis

This appendix shows the performance matrices for the Monte-Carlo analysis of TOPE estimators described in Section 5.4. This appendix is divided into four sets of figures; Figures D.1 through D.4 show the matrices for the comparison experiments performed upon the real world cost data visualised in Figure 2.1. Figures D.5 through D.8, Figures D.9 through D.12, and Figures D.13 through D.16 show the matrices for randomised simulated data using a fractal terrain generator with roughness parameters of $H = 0.2$, $H = 0.5$, and $H = 0.8$, respectively. Within each set there are four figures, showing the performance matrices for margins of 0%, 5%, 10%, and 25%.

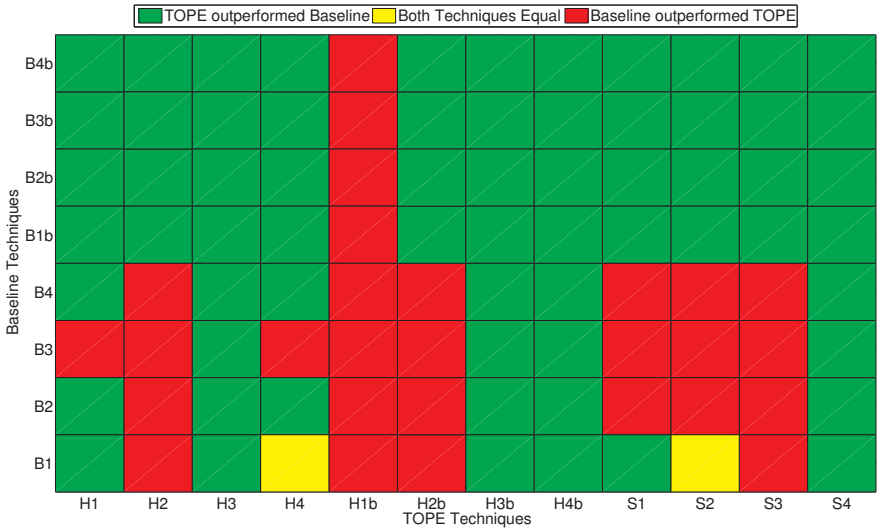


Figure D.1: Performance matrix comparing TOPE techniques against baseline techniques over real world data, with no performance margin.

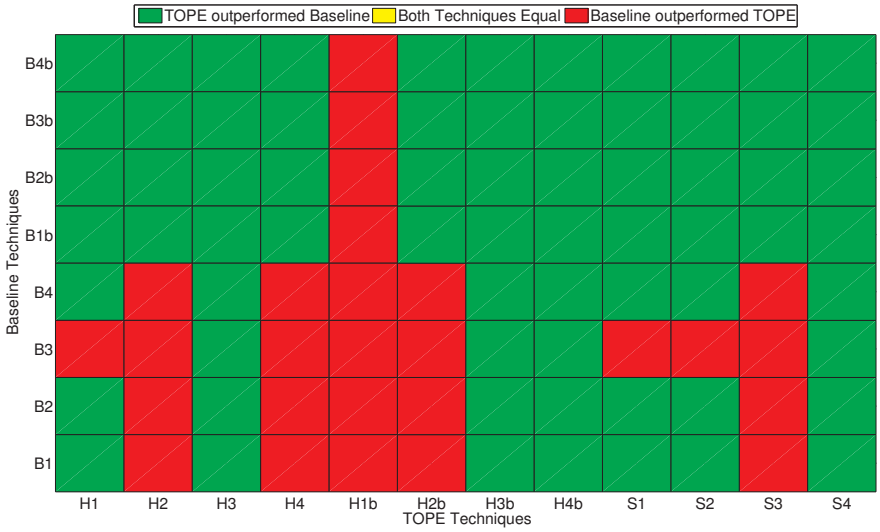


Figure D.2: Performance matrix comparing TOPE techniques against baseline techniques over real world data, with a performance margin of 5%.

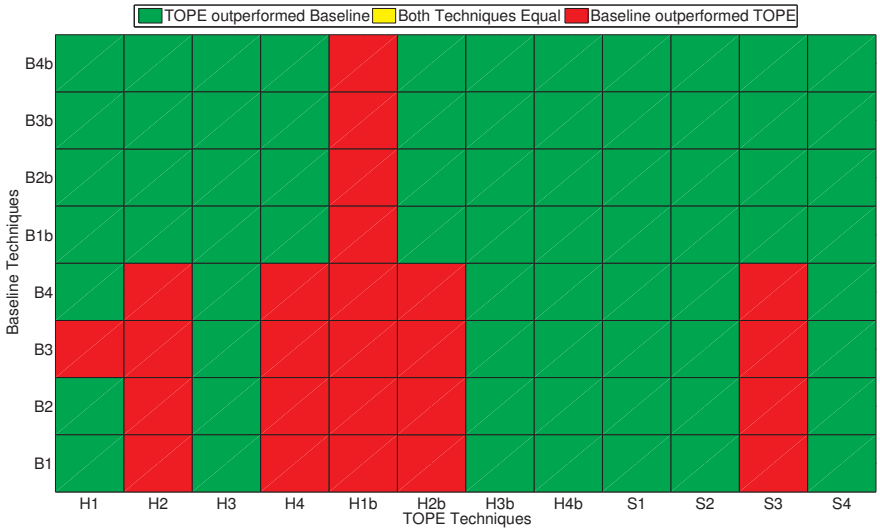


Figure D.3: Performance matrix comparing TOPE techniques against baseline techniques over real world data, with a performance margin of 10%.

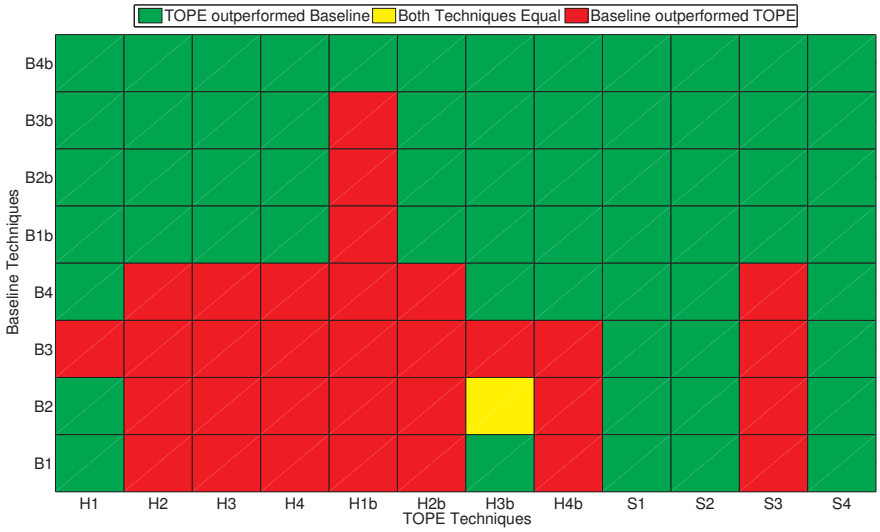


Figure D.4: Performance matrix comparing TOPE techniques against baseline techniques over real world data, with a performance margin of 25%.

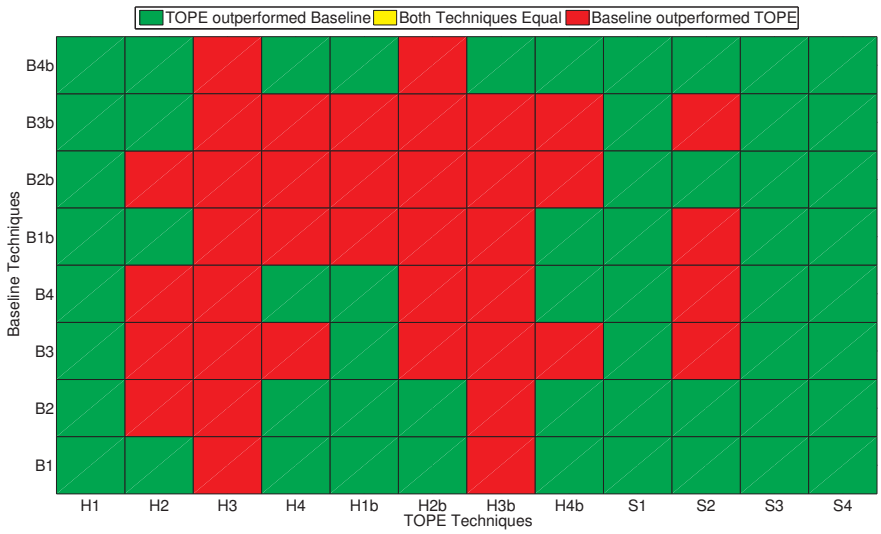


Figure D.5: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.2$, and with no performance margin.

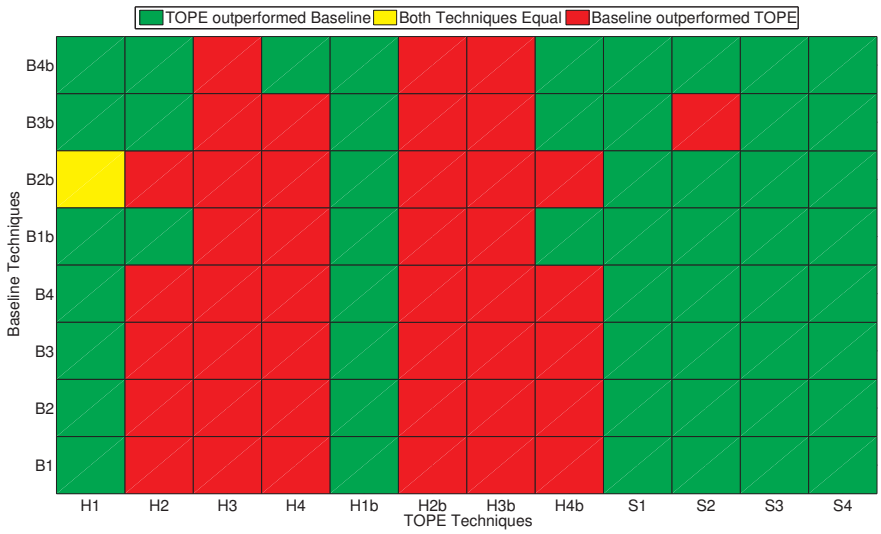


Figure D.6: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.2$, and with a performance margin of 5%.

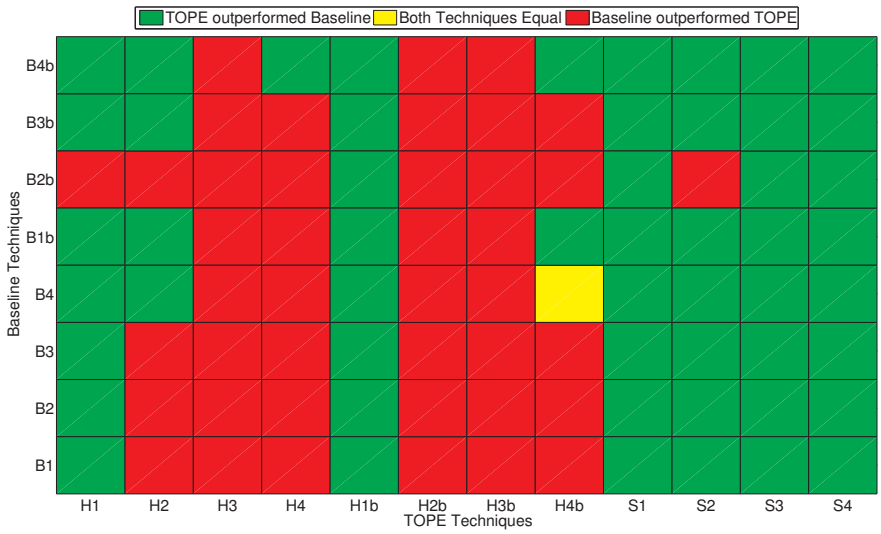


Figure D.7: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.2$, and with a performance margin of 10%.

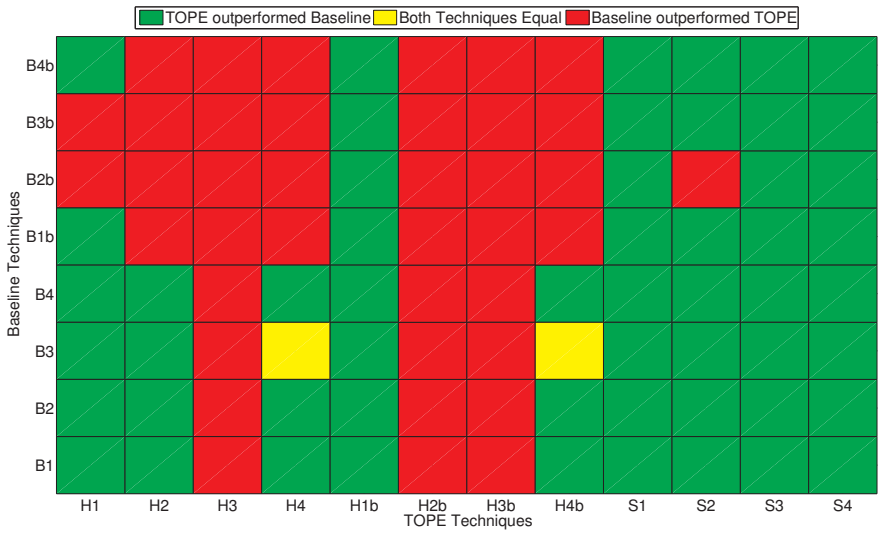


Figure D.8: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.2$, and with a performance margin of 25%.

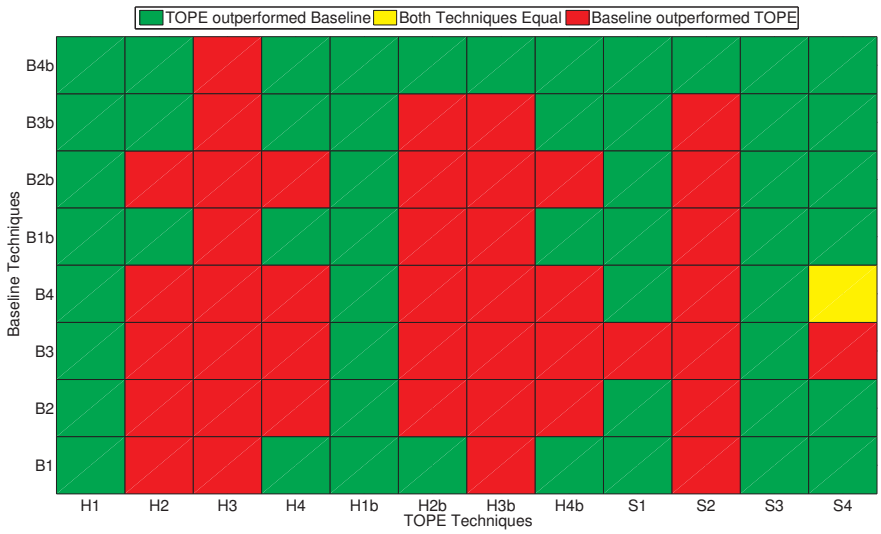


Figure D.9: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.5$, and with no performance margin.

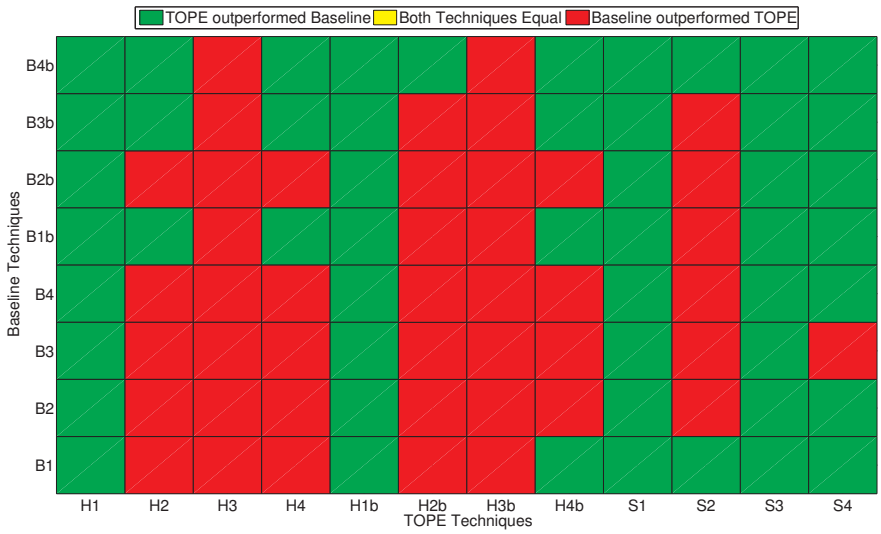


Figure D.10: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.5$, and with a performance margin of 5%.

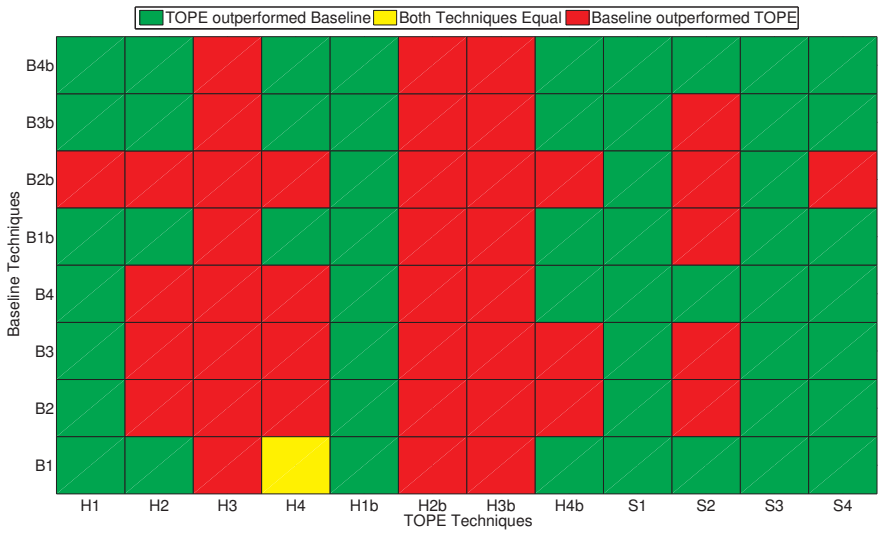


Figure D.11: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.5$, and with a performance margin of 10%.

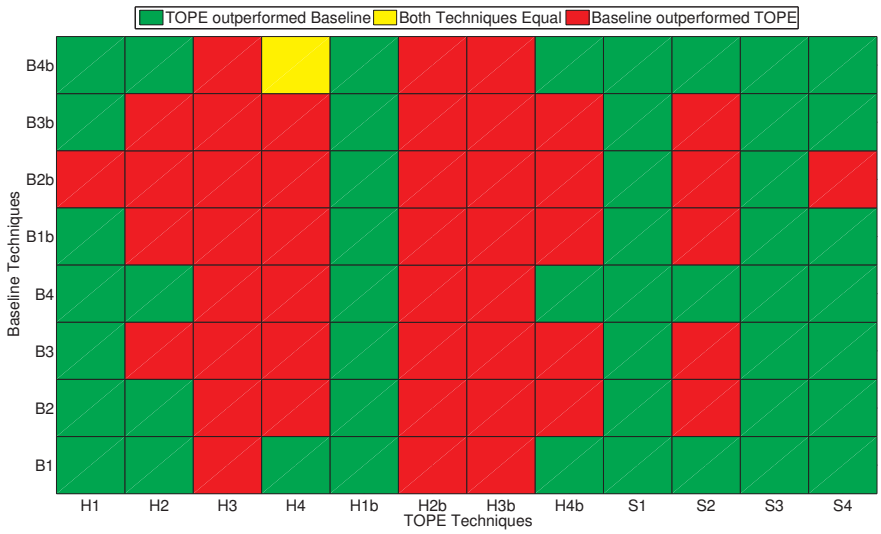


Figure D.12: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.5$, and with a performance margin of 25%.

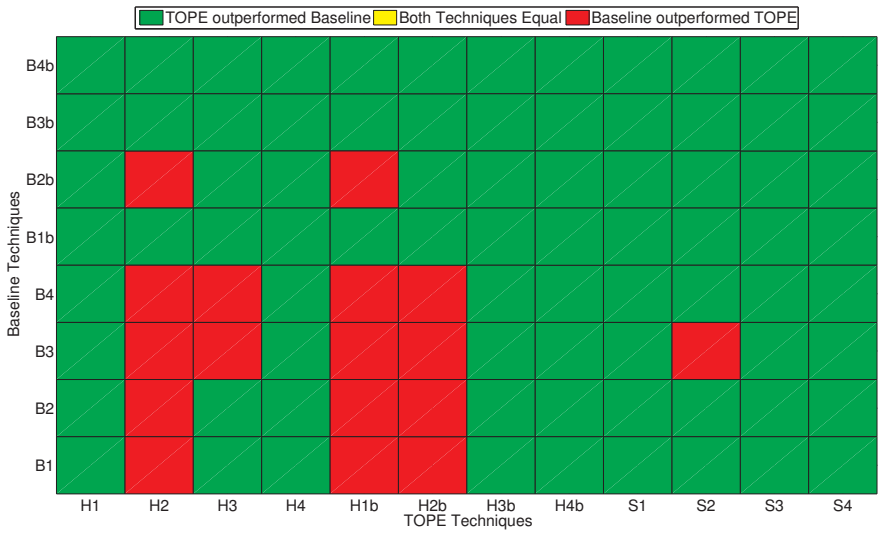


Figure D.13: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.8$, and with no performance margin.

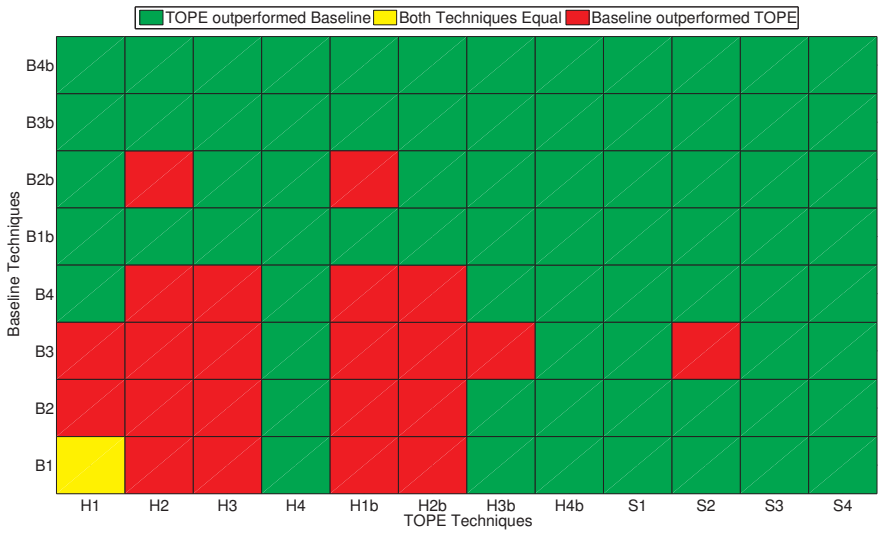


Figure D.14: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.8$, and with a performance margin of 5%.

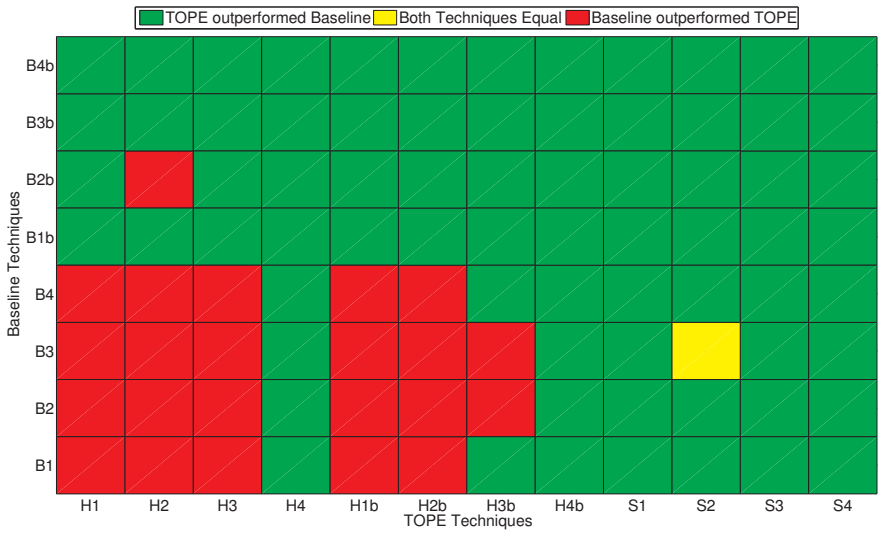


Figure D.15: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.8$, and with a performance margin of 10%.

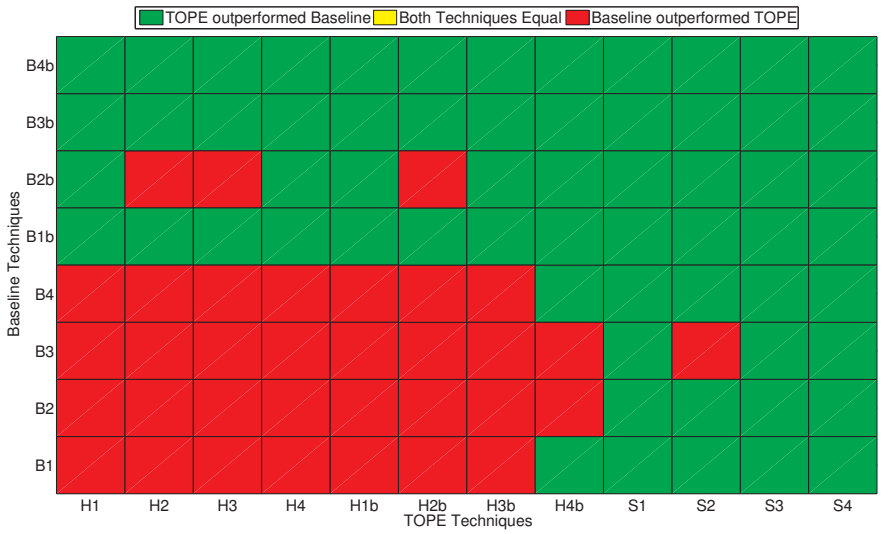


Figure D.16: Performance matrix comparing TOPE techniques against baseline techniques over randomised fractal terrain, with a roughness parameter of $H = 0.8$, and with a performance margin of 25%.

Bibliography

Allahverdi, A., Gupta, J. & Aldowaisan, T. (1999), 'A review of scheduling research involving setup considerations', *Omega* **27**(2), 219 – 239.

Allahverdi, A., Ng, C., Cheng, T. & Kovalyov, M. (2008), 'A survey of scheduling problems with setup times or costs', *European Journal of Operational Research* **187-3**, 985–1032.

Allen, T. (2010), Bounded anytime deflation, *in* 'Australasian Conference on Robotics and Automation', Brisbane, Australia.

Allen, T., Hill, A., Underwood, J. & Scheding, S. (2009), Dynamic path planning with multi-agent data fusion - the parallel hierarchical replanner, *in* 'Proceedings of the International Conference on Robotics and Automation', Kobe, Japan, pp. 3245–3250.

Allen, T., Underwood, J. & Scheding, S. (2007), A path planning system for autonomous ground vehicles operating in unstructured dynamic environments, *in* 'Australasian Conference on Robotics and Automation', Brisbane, Australia.

Amdahl, G. (1967), The validity of the single processor approach to achieving large-scale computing capabilities, *in* 'AFIPS Spring Joint Computer Conference', AFIPS Press, Atlantic City, N.J., pp. 483–485.

Andrews, G. (2000), *Foundations of Multithreaded, Parallel, and Distributed Programm*, Addison-Wesley.

Arjomandi, E. & Corneil, D. (1978), 'Parallel computations in graph theory', *SIAM Journal on Computing* **7**(2), 230–237.

Arkin, R. (1989), Towards the unification of navigational planning and reactive control, *in* 'AAAI Spring Symposium on Robot Navigation'.

Bekris, K., Chen, B., Ladd, A., Plaku, E. & Kavraki, L. (2003), Multiple query probabilistic roadmap planning using single query primitives, *in* 'IEEE/RSJ International Conference on Intelligent Robots and Systems'.

Belghith, K., Kabanza, F., Hartman, L. & Nkambou, R. (2006), Anytime dynamic path-planning with flexible probabilistic roadmaps, *in* 'IEEE International Conference on Robotics and Automation'.

Bellman, R. (1957*a*), *Dynamic Programming*, Princeton University Press, New Jersey.

Bellman, R. (1957*b*), 'A markovian decision process', *Journal of Mathematics and Mechanics* **6**, 679–684.

Blythe, J. & Scott-Reilly, W. (1993), Integrating reactive and deliberative planning for agents, Technical Report CMU-CS-93-155, Carnegie-Mellon University, Pittsburgh, PA, USA.

Bobrow, J. (1988), 'Optimal robot path planning using the minimum-time criterion', *IEEE Journal of Robotics and Automation* **4**, 443–450.

Boddy, M. & Dean, T. (1989), Solving time-dependent planning problems, *in* 'International Joint Conference On Artificial Intelligence'.

Borenstein, J. & Koren, Y. (1991), 'The vector field histogram fast obstacle avoidance for mobile robots', *IEEE Journal of Robotics and Automation* **7**(3), 278–288.

Brooks, R. (1985), A robust layered control system for a mobile robot, Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA.

Canny, J. (1988), *The Complexity of Robot Motion Planning*, MIT Press.

Carsten, J., Rankin, A., Ferguson, D. & Stentz, A. (2007), Global path planning on board the mars exploration rovers, *in* 'IEEE Aerospace Conference'.

Cerny, V. (1985), 'A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm', *Journal of Optimization Theory and Applications* **45**, 41–51.

Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2001), *Introduction to Algorithms*, 2 edn, MIT Press, Cambridge, MA.

Daly, J. (2010), There's millions in those milliseconds, *in* 'The Globe and Mail'. Available from `http://www.theglobeandmail.com/report-on-business/rob-magazine/theres-millions-in-those-milliseconds/article1443505/`.

Dean, T. & Boddy, M. (1988), An analysis of time-dependent planning, *in* 'Association for the Advancement of Artificial Intelligence'.

Dijkstra, E. (1959), 'A note of two problems in connexion with graphs', *Numerische Mathematik* **1**, 269–271.

Dirichlet, G. (1850), 'Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen', *Journal für die Reine und Angewandte Mathematik* **40**, 209–227.

Dreyfus, S. (1969), 'An appraisal of some shortest-paths algorithms', *Operations Research* **17**, 395–412.

Erickson, L. & LaValle, S. (2009), Survivability: Measuring and ensuring path diversity, *in* 'IEEE International Conference on Robotics and Automation'.

Ferguson, D., Likhachev, M. & Stentz, A. (2005), A guide to heuristic-based path planning, *in* 'Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)'.

Ferguson, D. & Stentz, A. (2006), 'Using interpolation to improve path planning: The field D* algorithm', *Journal of Field Robotics* **23**(2), 79–101.

Fiorini, P. & Shiller, Z. (1996), Time optimal trajectory planning in dynamic environments, *in* 'IEEE International Conference on Robotics and Automation'.

Fitch, R. & Butler, Z. (2008), 'Million module march: Scalable locomotion for large self-reconfiguring robots', *The International Journal of Robotics Research* **28**(3-4), 331–343.

Fournier, A., Fussel, D. & Carpenter, L. (1982), 'Computer rendering of stochastic models', *Communications of the ACM* **25**, 371–384.

Friedman, J., Bentley, J. & Finkel, R. (1977), 'An algorithm for finding best matches in logarithmic expected time', *ACM Transactions on Mathematical Software* **3**(3), 209–226.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1998), *Design Patterns*, Addison Wesley Longman, Inc.

Gancet, J. & Lacroix, S. (2004), Embedding heterogeneous levels of decisional autonomy in multi-robot systems, *in* '7th International Symposium on Distributed Autonomous Robotic Systems'.

Garvey, A. & Lesser, V. (1994), 'A survey of research in deliberative real-time artificial intelligence', *Real-Time Systems* **6**, 317–347.

Gauss, C. (1809), 'Theoria motus corporum coelestium in sectionibus conicis solem ambientium', Perthes, Hamburg. Translation reprinted as *theory of the motions of the heavenly bodies moving about the sun in conic sections*. Boston, Little, Brown and Company, Dover, New York, 1963.

Ghosh, S. (2007), *Distributed Systems - An Algorithmic Approach*, Chapman & Hall/CRC.

Grama, A., Gupta, A., Karypis, G. & V.Kumar (2003), *Introduction to Parallel Computing*, 2 edn, Addison-Wesley. ISBN 0-201-64865-2. Available from `http://www-users.cs.umn.edu/~karypis/parbook/`.

Grover, R. (2007), A model for sensing and reasoning in autonomous systems, PhD thesis, Department of Aerospace, Mechanical, and Mechatronic Engineering, The University of Sydney.

Gustafson, J. (1988), 'Reevaluating amdahl's law', *Communications of the ACM* **31**(5), 532–533.

Hammersley, J. & Handscomb, D. (1975), *Monte Carlo Methods*, Methuen, London.

Hansen, E. & Zhou, R. (2007), 'Anytime heuristic search', *Journal of Artificial Intelligence Research* **28**, 267–297.

Hansen, E. & Zilberstein, S. (2001), 'Montioring and control of anytime algorithms: A dynamic programming approach', *Artificial Intelligence* **126**, 139–157.

Hart, P., Nilsson, N. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics* **2**, 100–107.

Hart, P., Nilsson, N. & Raphael, B. (1972), 'Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"', *SIGART Newsletter* **37**, 28–29.

Heinzinger, G., Jacobs, P., Canny, J. & Paden, B. (1990), Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm, *in* 'IEEE International Conference on Robotics & Automation', Cincinnati, OH, pp. 150–155.

Horvitz, E. (1987), Reasoning about beliefs and actions under computational resource constraints, *in* 'Proceedings of the Third Workshop on Uncertainty in Artifical Intelligence', Seattle, WA, pp. 429–444.

Howard, T., Green, C., Kelly, A. & Ferguson, D. (2008), 'State space sampling of feasible motions for high-performance mobile robot navigation in complex environments', *Journal of Field Robotics* **25**, 325–345.

Hsu, D., Kindel, R., Latombe, J.-C. & Rock, S. (2002), 'Randomized kinodynamic motion planning with moving obstacles', *The International Journal of Robotics Research* **21**, 233–255.

Jarvis, R. (1984), Collision-free trajectory planning using distance transforms, *in* 'National Conference and Exhibition on Robotics'.

Jensen, R. & Veloso, M. (1998), Interleaving deliberative and reactive planning in dynamic multi-agent domains, *in* 'Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures', AAAI Press.

Joyeux, S., Alami, R. & Lacroix, S. (2007), A plan manager for multi-robot systems, *in* '6th International conference on Field and Service Robot Systems'.

Karumanchi, S. (2010), Off-road mobility analysis from proprioceptive feedback, PhD thesis, Department of Aerospace, Mechanical, and Mechatronic Engineering, The University of Sydney.

Karumanchi, S., Allen, T., Bailey, T. & Scheding, S. (2009), Non-parametric learning to aid path planning over slopes, *in* 'Robotics: Science and Systems Conference'.

Karumanchi, S., Allen, T., Bailey, T. & Scheding, S. (2010), 'Non-parametric learning to aid path planning over slopes', *International Journal of Robotics Research* **29**(8), 997–1018.

Kavraki, L. E., Svestka, P., Latombe, J.-C. & Overmars, M. H. (1996), 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces', *IEEE Transactions on Robotics and Automation* **12 (4)**, 566–580.

Kavraki, L. & LaValle, S. (2008), Motion planning, *in* B. Siciliano & O. Khatib, eds, 'Springer Handbook of Robotics', Springer, chapter 5, pp. 109–131.

Kelly, A. & Nagy, B. (2003), 'Reactive nonholonomic trajectory generation via parametric optimal control', *The International Journal of Robotics Research* **22**, 583–601.

Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., Happold, M., Herman, H., Mandelbaum, R., Pilarski, T., Rander, P., Thayer, S., Vallidis, N. & Warner, R. (2006), 'Toward reliable off road autonomous vehicles operating in challenging environments', *The International Journal of Robotics Research* **25**(5-6), 449–483.

Kirkpatrick, S., Gelatt, C. & Vecchi, M. (1983), 'Optimization by simulated annealing', *Science* **New Series 220**(4598), 671–680.

Kluge, K. & Morgenthaler, M. (2004), 'Multi-horizon reactive and deliberative path planning for autonomous cross-country navigation', *Unmanned Ground Vehicle Technology* **5422**, 461–472.

Knepper, R. & Kelly, A. (2006), High performance state lattice planning using heuristic look-up tables, *in* '2006 IEEE/RSJ International Conference on Intelligent Robots and Systems', pp. 3375 – 3380.

Knepper, R. & Mason, M. (2009), Path diversity is only part of the problem, *in* 'IEEE International Conference on Robotics and Automation'.

Knepper, R., Srinivasa, S. & Mason, M. (2010), Hierarchical planning architectures for mobile manipulation tasks in indoor environments, *in* 'International Conference on Robotics and Automation'.

Koenig, S. & Likhachev, M. (2002), D* Lite, *in* 'National Conference on Artificial Intelligence', number 18, AAAI Press, Menlo Park, CA, pp. 476–483.

Koenig, S. & Likhachev, M. (2005), 'Fast replanning for navigation in unknown terrain', *Transactions on Robotics* **21 (3)**, 354–363.

Koenig, S., Likhachev, M. & Furcy, D. (2004), 'Lifelong planning A*', *Artificial Intelligence Journal* **155 (1-2)**, 93–146.

Korf, R. E. (1987), Real-time heuristic search: First results, *in* 'Sixth National Conference on Artificial Intelligence'.

Kramer, J. & Scheutz, M. (2003), GLUE–a component connecting schema-based reactive to higher-level deliberative layers for autonomous agents, *in* 'Proceedings of FLAIRS2003', pp. 22–26.

Kuffner, J. & LaValle, S. (2005), An efficient approach to path planning using balanced bidirectional RRT search, Technical Report CMU-RI-TR-05-34, Carnegie Mellon University.

LaValle, S. & Kuffner, J. (1999), Randomized kinodynamic planning., *in* 'IEEE International Conference on Robotics and Automation', pp. 473–479.

LaValle, S. M. (2006), *Planning Algorithms*, Cambridge University Press, Cambridge, UK. Available from `http://planning.cs.uiuc.edu/`.

Lemons, S., Benton, J., Ruml, W., Do, M. & Yoon, S. (2010), Continual on-line planning as decision-theoretic incremental heuristic search, *in* 'AAAI 2010 Spring Symposium on Embedded Reasoning'.

Likhachev, M. & Ferguson, D. (2009), 'Planning long dynamically feasible maneuvers for autonomous vehicles', *The International Journal of Robotics Research*.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. & Thrun, S. (2005*a*), Anytime dynamic A*: An anytime, replanning algorithm, *in* 'Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)'.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. & Thrun, S. (2005*b*), Anytime dynamic A*: The proofs, Technical Report CMU-RI-TR-05-12, Robotics Institute, Pittsburgh, PA.

Likhachev, M., Gordon, G. & Thrun, S. (2003), ARA*: Anytime A* with provable bounds on sub-optimality, *in* 'Neural Information Processing Systems'.

Likhachev, M. & Stentz, A. (2008), R* search, *in* 'National Conference on Artificial Intelligence'.

Lumelsky, V. & Stepanov, A. (1986), 'Dynamic path planning for a mobile automaton with limited information on the environment', *IEEE Transactions on Automatic Control*.

Matthews, G. (2008), Asynchronous decision making for decentralised autonomous systems, PhD thesis, Department of Aerospace, Mechanical, and Mechatronic Engineering, The University of Sydney.

Miller, G. (1986), 'The definition and rendering of terrain maps', *Computer Graphics*.

Moravec, H. & Elfes, A. (1984), High resolution maps from wide angle sonar, Technical report, Carnegie Mellon University.

Murphy, R. (2000), *Introduction to AI Robotics*, MIT Press.

Murphy, R., Marzilli, A. & Hughes, K. (1997), When to explicitly replan paths for mobile robots, *in* 'IEEE International Conference on Robotics and Automation'.

Nise, N. (2004), *Control Systems Engineering*, 4 edn, Wiley.

OpenStreetMap Foundation (2010). See `http://www.openstreetmap.org`.

Papadimitriou, C. (1994), *Computational Complexity*, Addison Wesley.

Papadimitriou, C. & Tsitsiklis, J. (1987), 'The complexity of markov decision processes', *Mathematics of Operations Research* **12**(4), 441–450.

Pearl, J. (1984), *Heuristics*, Addison-Wesley.

Peynot, T. & Lacroix, S. (2006), Selection and monitoring of navigation modes for an autonomous rover, *in* 'International Symposium on Experimental Robotics'.

Phipps, M. & Quine, M. (1998), *A primer of statistics : data analysis, probability, inference*, 3 edn, Prentice Hall, Sydney.

Pirzadeh, A. & Snyder, W. (1990), A unified solution to coverage and search in explored and unexplored terrains using indirect control, *in* 'IEEE Internation Conference on Robotics and Automation'.

Pivtoraiko, M. & Kelly, A. (2005*a*), Efficient constrained path planning via search state lattices, *in* 'The 8th International Symposium on Artifical Intelligence, Robotics, and Automation in Space'.

Pivtoraiko, M. & Kelly, A. (2005*b*), Generating near minimal spanning control sets for constrained motion planning in discrete state spaces, *in* 'International Conference on Intelligent Robots and Systems'.

Pohl, I. (1970), 'Heuristic search viewed as path finding in a graph', *Artificial Intelligence* **1**, 193–204.

Rabin, S. (2003), *AI Game Programming Wisdom 2*, Charles River Media.

Rao, V. N. & Kumar, V. (2005), 'Parallel depth first search. part i. implementation', *International Journal of Parallel Programming* **16:6**, 479–499.

Rasmussen, C. & Williams, C. (2006), *Gaussian Processes for Machine Learning*, MIT Press.

Rezaei, S., Guivant, J. & Nebot, E. M. (2003), Car-like robot path following in large unstructured environments, *in* 'International Conference on Intelligent Robots and Systems', Vol. 3, pp. 2468–2473.

Rowe, N. (1990), 'Roads, rivers, and obstacles: Optimal two-dimensional path planning around linear features for a mobile agent', *International Journal of Robotics Research* **9**(6), 67–74.

Ruml, W. & Do, M. (2007), Best-first utility-guided search, *in* 'International Joint Conference on Artificial Intelligence', pp. 2378–2384.

Russell, S. & Norvig, P. (2003), *Artifical Intelligence: A Modern Approach*, 2 edn, Prentice Hall.

Russell, S., Wefald, E., Karnaugh, M., Karp, R., McAllester, D., Subramanian, D. & Wellman, M. (1991), Principles of metareasoning, *in* 'Artificial Intelligence', Morgan Kaufmann, pp. 400–411.

Russell, S. & Zilberstein, S. (1991), Composing real-time systems, *in* 'International Joint Conference on Artificial Intelligence', pp. 212–217.

Saitoh, T., Suzuki, M. & Kuroda, Y. (2010), 'Vision-based probabilistic map estimation with an inclined surface grid for rough terrain rover navigation', *Advanced Robotics* **24**, 421–440.

Samet, H. (1982), 'Neighbor finding techniques for images represented by quadtrees', *Computer Graphics and Image Processing* **18**, 37–57.

Samet, H. (1988), 'An overview of quadtrees, octrees, and related hierarchical data structures', *NATO ASI Series*.

Scheutz, M. & Kramer, J. (2006), Radic: a generic component for the integration of existing reactive and deliberative layers, *in* 'Fifth International Joint Conference on Autonomous Agents and Multiagent Systems', ACM, New York, NY, USA, pp. 488–490.

Schlemmer, M. & Agrawal, S. (2002), 'A computational approach for time-optimal planning of high-rise elevators', *IEEE Transactions on Control Systems Technology* **10**, 105–111.

Shiller, Z. & Dubowsy, S. (1991), 'On computing time-optimal motions of robotics manipulators in the presence of obstacles', *IEEE Transactions on Robotics & Automation*.

Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A. & Schwehr, K. (2000), Recent progress in local and global traversability for planetary rovers, *in* 'IEEE Conference on Robotics and Automation'.

Smith, D., Frank, J. & Jónsson, A. (2000), 'Bridging the gap between planning and scheduling', *Knowledge Engineering Review* **15**, 47–83.

Stentz, A. (1993), Optimal and efficient path planning for unknown and dynamic environments, Technical report CMU-RI-TR-93-20, Carnegie Mellon Robotics Institute.

Stentz, A. (1994), Optimal and efficient path planning for partially-known environments, *in* 'IEEE International Conference on Robotics and Automation', pp. 3310–3317.

Stentz, A. (1995), The focussed D* algorithm for real-time replanning, *in* 'International Joint Conferences on Artificial Intelligence'.

Stentz, A. & Hebert, M. (1995), A complete navigation system for goal acquisition in unknown environments, *in* 'International Conference on Intelligent Robots and Systems'.

Stewart, J. (1999), *Calculus*, Brooks/Cole Publishing Company.

Strandberg, M. (2004), Augmenting RRT-planners with local trees, *in* 'IEEE International Conference on Robotics & Automation', pp. 3258–3262.

Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, MIT Press.

Sykulski, A., Adams, M. & Jennings, N. (2010), On-line adaptation of exploration in the one-armed bandit with covariates problem, *in* '9th International Conference on Machine Learning and Applications', Washington DC, USA, pp. 459–464.

Tesauro, G. (1995), 'Temporal difference learning and TD-gammon', *Communications of the Association for Computing Machinery* **38**(3), 58–69.

Thayer, J. & Ruml, W. (2008), Faster than weighted A*: An optimistic approach to bounded suboptimal search, *in* 'Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling'.

Tompkins, P., Stentz, A. & Wettergreen, D. (2004), Global path planning for mars rover exploration, *in* 'IEEE Aerospace Conference'.

Tran-Thanh, L., Chapman, A., Munoz De Cote, E., Rogers, A. & Jennings, N. (2010), Epsilon–first policies for budget–limited multi-armed bandits, *in* 'Twenty-Fourth AAAI Conference on Artificial Intelligence', Atlanta, Georgia, USA, pp. 1211–1216.

Tsitsiklis, J. (1984), Problems in decentralised decision making and computation, PhD thesis, Massachusetts Institute of Technology.

Underwood, J. (2009), Reliable and safe autonomy for ground vehicles in unstructured environments, PhD thesis, Department of Aerospace, Mechanical, and Mechatronic Engineering, The University of Sydney.

Urstadt, B. (2009), Trading shares in milliseconds, *in* 'Technology Review'. Available from `http://www.stocktrading.com/mitsubpenny.pdf`.

Valve Corporation (2010), 'Steam hardware survey'. Available from `http://store.steampowered.com/hwsurvey/cpus/`.

Vasudevan, S., Ramos, F., Nettleton, E. & Durrant-Whyte, H. (2009), 'Gaussian process modeling of large scale terrain', *Journal of Field Robotics* **26 (10)**, 812–840.

Voronoi, G. (1907), 'Nouvelles applications des paramètres continus à la théorie des formes quadratiques', *Journal für die Reine und Angewandte Mathematik* **133**, 97–178.

Voss, R. (1987), Fractals in nature: Characterization, measurement, and simulation, *in* 'SIGGRAPH'.

Watkins, C. (1989), Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, England.

Yahja, A., Singh, S. & Stentz, A. (1998), Recent result in path planning for mobile robots operating in vast outdoor environments, *in* 'Symposium on Image, Speech, Signal Processing and Robotics'.

Yahja, A., Stentz, A., Singh, S. & Brumitt, B. L. (1998), Framed-quadtree path planning for mobile robots operating in sparse environments, *in* 'IEEE Conference on Robotics and Automation'.

Zelinsky, A. (1992), 'A mobile robot exploration algorithm', *IEEE Transactions on Robotics and Automation*.

Zilberstein, S. (1996), 'Using anytime algorithms in intelligent systems', *Artificial Intelligence Magazine* **17**, 73–83.

Zilberstein, S. & Russell, S. (1995), *Imprecise and Approximate Computation*, Kluwer Academic Publishers, Norwell, MA, chapter 4: Approximate Reasoning using Anytime Algorithms.