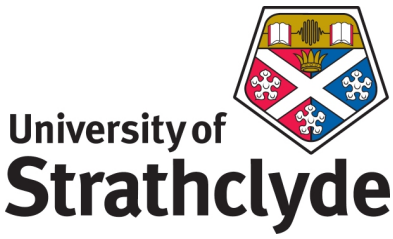
Online Generation and Use of Macro-Actions in Forward-Chaining Planning

Amanda Smith



A Thesis submitted for the degree of Doctor of Philosophy

Department of Computer and Information Sciences

University of Strathclyde

2007

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank Maria Fox not only for her supervision in my PhD work; but for her help in all matters academic and career related. My thanks also go to Derek Long for his support, in particular in the first year and in entering Marvin the 2004 planning competition.

Funding for the duration of the three years has come from two sources: the first year was generously supported by Maria Fox. The funding of the final two years was provided by the Department of Computer and Information Sciences at the University of Strathclyde.

Special thanks go to Andrew Coles for his insights and collaboration; as well as his endless personal support, company and care. Thanks also to colleagues past and present, who have made the last three years much more enjoyable than they otherwise would have been: Alastair, Alex, John, Jonathan, Henrik, Keith, Newton, Nor, Pete, Richard and Stephen.

Thanks to Malte Helmert for making the source code for the downward planner available for use in integrating the Causal Graph heuristic into Marvin.

Finally I dedicate this thesis to my parents, with thanks for a lifetime of support and encouragement, to bring a positive end to what has been a difficult year.

Abstract

This thesis presents a technique for online learning and management of macro-actions in forward chaining planning. Macro-actions are learnt on plateaux, areas of the search landscape where the heuristic cannot offer good search guidance, and are reused when future plateaux are encountered. Libraries of macro-actions are generated, storing macro-actions for use on future problems. Several strategies for the management and pruning of such libraries are considered allowing the planner to maintain a smaller collection of macro-actions to minimise potential negative effects on performance.

The work is extended to investigate the potential for the simulation of macro-actions without increasing the branching factor during search. Actions are re-ordered during enforced hill-climbing (EHC) search, and best-first search, based on the number of times they have followed the action currently at the tail of the plan in past solutions. In EHC this is equivalent to suggesting two-step macro-actions without increasing the branching factor.

The techniques are evaluated across a wide range of domains with differing properties under the relaxed planning graph heuristic. The results show that a library of plateau-escaping macro-actions can improve planner performance using only online learning techniques to select the best macro-actions to keep. Search time pruning is shown to be very effective; whilst pruning the library of macro-actions based on the number of times each action is used can offer further improvements. Action reordering has been shown to offer performance improvements, albeit not as great as those produced by using macro-actions, but without the occasional degradation in performance on some problems that is often associated with macro-actions.

Chapter 1

Introduction

1.1 Domain-Independent Planning

Domain-independent planning (hereafter referred to as planning) is a P-SPACE hard search problem [19]; the aim of which is to find a series of actions that can transform an initial state into a desired goal state. More formally an instance of a planning problem can be defined as a tuple <A, I, G> where A is the set of actions that can be applied in the domain, I is a set of propositional formulæ defining the initial state and G is a set of propositional formulæ defining the goal state. The problem is to find an ordered sequence of actions from A that transforms the initial state, I, into a state S such that $G \subseteq S$. Each action in A is made up of a set of preconditions, a set of propositional formulæ describing the conditions that must hold for the action to be applied, and a set of effects consisting of those propositional formulæ that become true once the action has been executed (formulæ become false by virtue of their negations being made true).

The set of actions A is referred to as the domain of the problem; this encompasses the definition of predicates in the world: the predicates to be used are those that occur in the preconditions or effects of the actions. The predicate sets I and G refer to the specific instance of a problem for a given domain. Many different instances of a planning problem can be posed, with the same action set, A, and different initial and goal states, I and G. These are referred to as different *problems* in the same *domain*.

The flexibility of planning arises from the fact that the problem definition allows for abstraction; many different problems can be modelled as planning problems and solved using a single system, a *planner*. As a result of this flexibility, planning has been widely researched both as an intersting theoretical problem and

as a practical tool for solving a variety of problems. The language used to express planning problems has developed as technologies for solving these problems have improved. Initially planning problems were defined using the STRIPS formalism [20] which allowed states to be defined as conjunctions of predicates. This was later extended to ADL [62] to allow greater expressivity for defining problems, including allowing the use of negative and disjunctive preconditions and universally quantified effects. The planning domain definition language, PDDL [54], was developed to provide a standard format in which planning domains, using the ADL and STRIPS formalisms, could be specified.

The expressiveness of the language has been further developed to allow expression of temporal and numerical constraints [22], derived predicates [18, 35] and more complex temporal requirements [25]. Further extensions have also been made to allow the expression of uncertain and probabilistic effects, such as those used in the probabilistic track of the Fourth International Planning Competition [72]. The scope of the work in this thesis is within 'classical planning' which includes the ADL language extensions but does not include domains with temporal and numerical resources, although the work could be extended to include domains with these features. Classical planning makes the assumptions that the world is unchanging and the outcomes of actions are determinisic.

1.2 Search in Planning

A planning problem can be thought of as a search problem in which the states are the world states and the application of an action generates the successors of a state[1]. Exhaustive search is not an effective means of solving any but the smallest of planning instances due to the complexity of the problem. In order to successfully solve interesting planning problems a guided search must be used. Forward heuristic search has become a very popular technique for solving planning problems.

1.3 Additional Search Guidance

Planning with generic heuristics has enabled modern planning technologies to solve much larger problem instances than planning through exhaustive search.

[1]This view is taken throughout this thesis although other views have been taken, such as modelling the problem as a search through plan-space as in partial-order planners such as VHPOP[38] and modelling planning problems as SAT instances such as in Blackbox [40]

It still remains, however, that modern planning technology needs further improvement if it is to solve useful real-world problems. Whilst the approach of improving current heuristics to provide better generic heuristics for search is one area of research; the development of other ways to allow the planner to search more efficiently is also being researched. A large area of this research is dedicated to allowing the planner to learn, through exploring a number of problem instances before solving some other, usually larger instances of the problem. The disadvantages of the learning approach are the time required for the learning process to take place and the fact that the, usually small, training instances may not be representative of the larger problems the planner is later asked to solve. This affects the ability of the planner to learn useful information from the training problems. Other approaches make use of domain-specific knowledge, which is often hand-coded, and is given to the planner as additional information with the domain [1].

Macro-actions have been used in planning for quite some time as a means to improve search performance. A macro-action is an action added to the domain in order to encapsulate sequences of actions which will often occur in solution plans. The effects of applying a macro-action are the same as the effects of applying its constituent actions a specific order. In previous work, macro-actions have been either learnt off-line on a number of training problems before search begins and then added to the domain [8]; or have been hand coded and given to the planner as part of the domain [57].

1.4 Statement of Thesis

The thesis presented in this work is that an online learning strategy for generating and managing a library of macro-actions improves the performance of FF, one of the most popular state of the art planners, across a wide range of domains. This is demonstrated to be effective under the two most powerful generic planning heuristics. Macro-actions are extracted from places in the search space where the heuristic fails to guide search. A further contribution is a technique to simulate the benefits of macro-actions by dynamically altering the order in which the planner considers actions, weakly suggesting macro-actions without increasing the branching factor of search. This technique offers significant improvement in planner performance.

1.5 Document Overview

The remainder of the document is structured as follows:

- **Chapter 2** Puts the work in context detailing past domain-independent planning systems and previous work done on learning macro-actions in planning.
- **Chapter 3** Describes the planner Marvin that competed in the Fourth International Planning Competition. The core features are discussed and the macro-action learning techniques are introduced.
- **Chapter 4** Details further contributions of the work discussing the macro-action library management and action reordering strategies used.
- **Chapter 5** Gives a full evaluation of the techniques introduced in chapter 4 presenting results across a varied selection of evaluation domains.
- **Chapter 6** Details the conclusions made and gives a summary of the work presented. It concludes with some potential future directions for extending the work presented.

Chapter 2

Background

The work presented in this thesis considers learning and using macro-actions in forward-chaining planning and the management of a library of generated macro-actions. This chapter details work already done in the field beginning with an overview the development of planning technology. It then goes on to detail more-recent planners using forward heuristic search. Work done on using macro-actions in planning is reviewed. The chapter concludes with an overview of the relevant areas of learning in planning that are related to online learning and the management of a library of macro-actions.

2.1 Early Approaches to Planning

This section gives a brief overview of planning. Initially planning started as state space search with the STRIPS [20] planner. It later developed into a search through partial plans. Following partial-order planning, most planning research divided into three distinct directions.

2.1.1 STRIPS

One of the earliest planning systems was the Stanford Research Institute Planning System (STRIPS) [20]. The formalism used for describing actions in this planner is still widely used today. STRIPS used a domain based on a robot which had to move between rooms and carry out the task of moving boxes around. The planning system itself used a regression search technique starting from the goal and working backwards to find a plan. An operator that achieves a required goal is selected for application. The preconditions of the chosen operator are added to the stack of goals to be achieved. Goals are popped off the goal stack and, if

the goals are not already achieved, actions achieving these goals are selected to achieve the goals, with their preconditions being added to the goal stack. Search terminates when the goal stack becomes empty.

2.1.2 Early Partial-Order Planners

Planning can be thought of in many different ways. One approach is to treat the problem as a search through a space of partially complete plans [71]; this was the approach taken by many early planners. The search begins with a plan consisting of a dummy start action, the effects of this action being the predicates that are true in the initial state; and an end action, the preconditions of which are the goal conditions imposed by the problem. The aim of the search is to find a complete plan that links these two states such that the each precondition of every action is linked to an effect of another action, by a *causal link*, and no action between the achieving effect and the precondition it achieves threatens the causal link, i.e. deletes the required effect.

Search proceeds by repeatedly selecting a precondition to achieve next, inserting an action into the plan to achieve it and resolving any threats the new action may pose to existing causal links. Different partial-order planners use different strategies for deciding the order in which to satisfy preconditions, and also for deciding which threats to causal links to resolve first. A simple example of a strategy is first-in-first-out (FIFO) considering the threat that was created the earliest for resolution first.

When inserting a new action in to the plan it is necessary to ensure that all the existing causal links in the plan are protected. Suppose action a_1 adds predicate p_1 in the plan, which is later required as a precondition of action a_2. A causal link will exist between the effect p_1 of action a_1 and the precondition p_1 of action a_2. If an action a_3 that deletes p_1 is now added to the plan to achieve some unsupported precondition in the plan p_2 then the existing causal link is threatened. This can be resolved in one of two ways, by *promotion* or *demotion*. Promotion is the addition of an ordering constraint to the plan showing that action a_3 must happen before a_1; conversely demotion is the addition of an ordering constraint showing that action a_3 must occur after a_1. When planning with lifted actions (i.e. actions whose parameters have not yet been bound to specific entities, but instead remain as variables) there is an additional possible way to proceed, *separation*. This involves binding the problematic parameter of a_3 to a different object than that used in p_1. For example, suppose the threatened precondition was (at truck1 location1) and an action (drive ?truck ?location location2), with effect (not (at

?truck ?location)), was added to the plan[1]. To resolve the potential problem that this action may delete (at truck1 location1) it is possible to impose a *binding constraint* stating that ?truck cannot be bound to truck 1.

The advantage of this approach over that of searching through state space is that the plans generated are more flexible for execution. The completed plans are only partially ordered, that is, the only ordering constraints imposed by the planner are those that are required for the plan to remain sound. No arbitrary ordering decisions are made regarding the order in which the plan must be executed: this provides a much more flexible plan for an executive to work with. The problem with partial order planning is search control: deciding the order in which to satisfy the remaining unsatisfied preconditions and which achievers to use to do so is crucial to finding a plan efficiently. Heuristics available to do this still do not allow the same level of performance achieved by other planning technologies.

NOAH and Nonlin

NOAH [66] was an early exponent of this style of planning. A plan is made to achieve each of the predicates in the goal state using means-ends analysis. These plans are then placed alongside each other in a partial-order plan structure. The planning process continues by making refinements to this abstract plan in order to transform it into a plan that can be executed in the real world. In achieving one goal predicate it is often the case that the achievement of another goal predicate is made impossible: this is a characteristic of non-linear problems. For example, if the goal is to be wearing socks and shoes, then putting one's shoes on before one's socks would mean that the overall goal could not be achieved without first destroying the goal of wearing shoes. In combining plans for individual goals it is therefore necessary to resolve any such conflicts to refine the plan to be a correct plan to achieve the complete conjunction of goals.

NOAH does this by viewing the individual plans as a single plan, with flaws, to solve the overall problem. Each flaw must be considered and resolved in order to complete an ordered plan. The planning process is a series of refinements of an existing sequence of actions: in this way the approach shares some common problems with planning with macro-actions: any actions that are to be considered for application simultaneously with a macro-action will require similar considerations.

[1] parameters marked with question marks are unbound variables that can take the value of any object in the problem

Partial-order planning was further extended through the development of the planner Nonlin [67]. Nonlin was an improved version of NOAH written with the intention of improving generating plans with fewer conflicts through the introduction of dependency-directed backtracking.

Tweak and UCPOP

The work on Tweak [11] formalised the partial order planning paradigm defining the concepts that were being used in partial order planning; such as declobbering, promotion and demotion. UCPOP [63] built on this work producing a widely available implementation of a partial order planner. It added several features allowing the use of more complex language constructs such as disjunctive preconditions, conditional effects and universal quantifiers. UCPOP was also the first partial-order planner to deal with lifted action schemata, allowing separation to be used to resolve conflicts.

Revival of partial order planning

REPOP [60] was written with the goal of reviving partial order planning to recover the advantage of having the flexible solution plans it generates. It offered a number of improved heuristics in an attempt to address the weaknesses of partial-order planning: inefficiency due to poor search guidance. The distance to the goal from a given plan is the number of actions that must be added to that plan in order to achieve a solution plan. In order to compute this value accurately, however, it would be necessary to first solve the planning problem; this is therefore not useful as a heuristic to find a solution. Earlier heuristics had, however, attempted to estimate this cost, for example by counting the number of preconditions yet to have an achiever in the plan (note that this is only an estimate since one precondition may require more than one action to achieve it; or two preconditions may be satisfied by the same action). To account for these interactions a heuristic based on the planning graph structure [4] was used. Other techniques were introduced to allow the planner to resolve conflicts more efficiently: one to use disjunctive constraints to protect threatened links and another using a reachability analysis to find unsafe links which had not yet been detected.

Recently the partial-order planning strategy has been continued by VHPOP [38] using sophisticated techniques for selecting the threats and open preconditions to resolve next and the relaxed planning graph heuristic to guide search—a heuristic used by many of the current state-of-the-art planners.

2.1.3 Hierarchical Planning

Hierarchical planning is based on the idea of creating an abstract plan and refining the details of the plan until the plan reaches a sufficient level of detail to be executed. The abstraction is linked to the idea of macro-actions. Macro-actions are one sort of hierarchical structure, they abstract based on actions, in a similar manner to the tasks in SHOP [57]. Some approaches abstract based on predicates in the domain [41] and some based on objects [3].

ALPINE

Knoblock created the ALPINE system [41] to automatically generate abstraction hierarchies for planning problems. Hierarchical structure can be a useful guide in solving planning problems. Much work on using hierarchical structure in solving problems requires human-encoding of the hierarchies: thus not adhering strictly to the principle of a domain independent system that can solve a variety of problems without human intervention. The ALPINE system is a fully automated system for generating such hierarchies; Prodigy [10] was then used to solve the problems using these hierarchies.

The system begins with the domain and optionally the problem definition. If a problem definition is supplied, a directed graph is built starting from the goal literals with a directed edge placed from all preconditions of actions that achieve each goal literal to the goal literal the action achieves. A directed edge is also created from each effect of the action to all other effects of the action. The edges indicate that the literals at the start of the edge must appear in the same, or a higher, abstraction level. Any connected components within these graphs, those in which all literals are connected to each other as part of a cycle, form an abstraction level. These abstraction levels are partially ordered by the precedence given in the original graph. A total order is generated from this partial order to form the abstraction levels for the given problem. The version using just the domain definition, and no problem specification, is less successful: it works in the same way, but instead of beginning building the graph from the goal conditions it considers all literals added by any action in the domain. In general this results in fewer abstraction layers, and therefore a less useful problem abstraction.

The hierarchies are used in search guidance by first solving the problem at the highest abstraction level and then moving to the next (lower) abstraction level and attempting to extend or repair the plan to solve the problem at that level. If it is not possible to modify the plan to work at the next level a different abstract plan must be created at the higher levels. A solution plan is one which solves the

problem at the lowest level of abstraction. Abstract hierarchies were shown to give an overall performance improvement when used to solve problems and make solving problems more time-efficient than when using control rules in Prodigy.

SHOP

SHOP [57] was written with the intention of creating a domain-independent hierarchical planner. It is still not a fully automated system since the higher-level tasks must be hand-coded for each domain that must be solved. For each domain a number of *methods* are defined. These methods consist of a number of lower level tasks that must be performed in order to complete a higher-level task; in this way they are similar to hand-coded macro-actions: they contain action sequences which are reasoned with at a higher level. Methods can contain a number of different ways of achieving the task, the preference order of which is determined by the order in which the preconditions appear. The planner initially reasons with these higher level concepts in order to solve the problem and then gradually breaks them down to primitive actions.

An example of a hierarchical task, considered in [57], is a domain that requires a person to travel from one place to another. There may be many methods of transport available to the person: they may travel by bus, by taxi or may simply walk. Each of the tasks breaks down into several other steps: walking may involve finding the route on a map, then following that route to the destination; getting a taxi may involve hailing the taxi, telling the driver the destination and paying for the taxi; getting the bus may involve walking to the bus stop, hailing the bus, buying a ticket and then leaving the bus at the required stop. Furthermore, an ordering may be placed on the preferred method of travel: walking may be preferred as a first choice, but only if the weather is reasonable; followed by getting a taxi; and the bus may only be used if there are insufficient funds for a taxi.

The search strategy is a forward search approach, the planner is given a task to complete instead of a goal set to satisfy. Methods that can be used to perform these tasks are selected; these are then decomposed using further methods and eventually split into primitive operators. Hierarchical planning was shown to have a number of advantages: the expressive power available was much greater than other planners as numerical expressions could be evaluated during planning time and arbitrary additional processing steps could be added in the operator expansion phase. It was also shown to outperform the best planners of the time although unfortunately it not possible to really make a fair comparison because

the hierarchical planner is given a lot of additional information about the planning domains through the construction of these tasks; even other planners using control rules, such as TL plan [1], cannot be used for direct comparison because the control rules each is given are different [65]. The major disadvantage of this approach is the requirement of hand-coded task hierarchies for each domain.

SHOP 2 [58] was an extension of SHOP that allowed two hierarchical tasks to be parallelised: in SHOP two hierarchical tasks cannot be interleaved. SHOP 2 extended the decomposition reasoning to allow temporal planning.

2.2 Early Approaches to the use of Macro-Actions and Learning From Previous Planning Experience

In this section previous approaches to the use and generation of macro-actions are discussed. Early general problem solving systems exhibited poor performance as they used very weak general heuristics to guide search. Such systems were revived by the use of learning techniques to allow better search guidance to be learnt.

2.2.1 Introduction to the use of Macro-Actions in Planning

The use of macro-actions in planning has been widely explored. Macro-actions consist of an ordered sequence of actions taken from the planning domain. The motivation for using macro-actions is intuitive: if a sequence of actions occurs many times in solution plans it is logical to suggest to the planner that this may be a good action sequence to consider. Indeed planning domains often consist of many macro-actions implicitly: each time a domain is created a certain level of abstraction must be decided on for each action: for example, an action to load a package onto a truck (as used in the driverlog domain from the third international planning competition [49]), is implicitly a macro-action for picking up the package, moving to the truck, and placing the package on the truck, indeed even these actions could be further broken down. In the context of a planning problem the single unit actions are considered to be those that are specified in the domain; macro-actions are sequences of one or more of these actions.

A more formal definition of a macro-action follows: pre_X, add_X and del_X are the sets of preconditions, add effects and delete effects of an action X re-

spectively. A macro-action of length 1 is any action taken from the domain. An action a_1 can be added to the end of an existing macro-action a_m if, and only if, $del\_a_m \cup pre\_a_1 = \emptyset$. The result of adding a an action a_1 to the end of a macro-action a_m is an action a_n where:

$pre\_a_n = pre\_a_m \cup pre\_a_1 \setminus (add\_a_m \cap pre\_a_1)$

$add\_a_n = add\_a_m \cup add\_a_1$

$del\_a_n = del\_a\_m \cup del\_a_1$

Using macro-actions when planning gives the potential for increased efficiency in solving problems as many steps can be planned by the application of one macro-action, thus avoiding search that would otherwise have to be done. There is, however, a cost associated with this potential gain; that is, the increased branching factor at each action choice point in the search: in addition to considering all possible groundings of the actions in the domain, all possible groundings of all macro-actions must also be considered.

The number of instantiations of an action is exponential in the number of parameters. Macro-actions consisting of two or more actions include the necessary parameters of their constituent actions. In general macro-actions have many more preconditions than the other actions in the domain, worst case the sum of the parameters of all the constituent actions (although this is rare since macro-actions usually have some actions with shared parameters), and are therefore expensive to instantiate.

For a planning problem with solution length l and number of possible action instantiations (branching factor) b the number of nodes expanded to solve the problem, in the worst case, will be O (b^l). If m macro-actions are added to the domain, with n being the total number of possible groundings of all of the macro-actions, the worst case complexity is O $(b+n)^l$. It is, however, important to note that if a macro-action is used in solving the problem then the depth, l, to which the search space needs to be explored (i.e. the plan length) is decreased meaning that if a macro-action of length k is used the number of nodes expanded will be reduced to O $((b+n)^{l-(k-1)})$.

Further problems arise regarding solution plan length, which is often a metric used to determine plan quality. If macro-actions are used it may well be the case that shorter action sequences could have been identified by search; this means that macro-actions must be carefully managed to ensure that they have minimal impact on plan length. It is therefore important, when considering evaluating a system using macro-actions, to consider not only the change in time taken to find

a plan; but also the effect on plan quality.

2.2.2 Macro-Actions in STRIPS

The STRIPS planner was modified to consider final solution plans as macro-actions [20, 21]. The action parameters in the plans were lifted and the plan was considered for use, in whole or in part, by the planner in future problem solving. The STRIPS approach quickly leads to a large number of macro-actions; despite this the authors demonstrated that it can show performance improvements in a number of domains: in the absence of an effective heuristic any guidance given to the search becomes much more important. This means that although the size of the search space was increased by the existing macro-actions, the value of the search guidance gained by using macro-actions was much greater.

2.2.3 REFLECT

The system REFLECT [15] used a preprocessing step in order to infer information about each domain before attempting to solve problems in that domain. The information inferred was of two types: identification of predicates which are statically mutex in the domain; and the generation of macro-actions from the domain description. Previous approaches to macro-action generation, such as that used in STRIPS, had extracted macro-actions from the solutions to previous problems. The approach taken in REFLECT is to combine operators from the domain without the requirement to solve any other problems, thus avoiding the action sequence choices being made based on incorrect search trajectories unwisely selected as a result of poor search guidance.

The algorithm used to generate the macro-actions capitalises on the relatively small size of the domains at the time. First all possible pairs of actions are considered as macro-actions. All actions pairs which have conflicts in preconditions and effects are discarded; in practice (in the domains used) this removes a lot of the possible combinations. Further pruning is done by allowing only those operators that can have a common parameter binding to be kept. Due to the small number of actions in the domains used by the planner this leaves a small number of macro-actions (three in the blocksworld domain) to be used in solving the problem.

2.2.4 MORRIS

In 1985 Minton [56] noted that better search performance could be achieved if fewer macro-actions were memoised and that many of the macro-actions memoised by STRIPS could never potentially be useful. MORRIS extends the STRIPS approach by introducing a strategy to prune the macro-actions generated in order to gain the benefit of using macro-actions without the search space becoming too large. Two types of macro-actions were identified, S-macros, those which are used frequently in many solutions and T-macros, those which can assist in places where the heuristic causes search to proceed in the wrong direction and where goals are tackled in a sub-optimal order. MORRIS uses these macro-actions in a best-first search strategy. The results showed that the T-macros offered the most significant saving in terms of time taken and nodes explored; while the S-macros were more often used but did not offer a great deal of performance improvement.

2.2.5 MACLEARN

The MACLEARN system [39] learns macro-actions to be used in solving planning problems. The system consists of three parts a macro-action proposer, a static macro-action filter and an dynamic macro-action filter. Macro-action learning, via the macro-action proposer, is triggered during planning when a peak is detected in the heuristic value[2]. Peaks occur when both the successor state and previous state have a worse (lower) heuristic value than the state currently being expanded. When a peak has been detected the actions since the last peak are used to form a macro-action (if no previous peak is found the previous peak is taken to be the root of the search tree). The motivation for this approach is that good macro-actions should remove valleys from the heuristic landscape.

The static macro-action filter removes macro-actions which can trivially be found to be of little use in solving the problem. Firstly, all macro-actions that are simply primitive operators in the original domain are removed. Secondly, all macro-actions above a specified length are removed. Finally, a domain dependent macro-action pruning strategy can be specified and given to the planner to further prune the number of macro-actions. The dynamic filter removes from the list any macro-actions that are not used in solving problems. The dynamic filter is manually invoked to avoid the problem that some useful macro-actions may be pruned when the filter is ran simply because they have recently been discovered and few, or no, problems have yet been solved since their generation.

[2]note here that a large heuristic value is considered to represent a good state rather than the usual convention in planning of a smaller heuristic value representing a good state.

2.2.6 Macro Problem Solver

Korf's Macro Problem Solver [43] arose from the idea of planning for individual goals, with a fixed goal ordering, in order to achieve a collection of dependent subgoals. Problem solving occurs in two phases: first the macro-action generation step generates a collection of macro-actions to be used in solving all problems of a given type from any valid initial state to a single fixed goal state; the solution phase then uses these macro-actions to solve the problem.

In order to understand how the macro-actions are generated it is helpful to first understand the nature of the macro-actions and how they are used. The macro-actions generated in the generation phase are collected into a table. Each row in the table represents the possible states of objects and the column represents the object that is in that state. For example, in the sliding tile 8 puzzle the columns correspond to the tiles in the problem and the rows correspond to each possible position in the grid they can be placed. Entries in the table are the macro-action which will move the tile from the position indicated by the row number to the (fixed) goal position for that tile without altering the position of any higher priority goal. The priority of goals is decided before attempting to solve the problem and simply indicates an ordering in which goals should be achieved: higher priority goals are achieved first. Macro-actions are defined such that the complete execution of any macro-action in the table will not destroy any goals at a higher level than the goal they are designed to achieve. Goals may temporarily be destroyed during the application of the macro-action but will be re-instated before the execution of the macro-action is complete. For example, in the eight-puzzle, with the goal ordering placing the 'blank' square first and then the tiles in the order 1 to 8, the macro-action to move tile 4 from any position on the board to its goal location will guarantee that the tiles 1, 2, 3 and blank will be in their goal position after it has been executed (assuming that they were in their goal positions before execution), even though they may be moved in the process of executing the macro-action.

The properties of the macro-actions in the macro-action table turns the solution phase of the problem into simply a number of lookups in a table, the system first applies the macro-action to achieve the highest priority goal in the system by transforming the object concerned from its current position, then the next goal in the priority list from its position following the application of the first macro-action and so on until all goals are satisfied. The result of this approach is that all search is removed from the actual solution phase of the problem. The requirement to solve subgoals separately whilst leaving higher priority goals un-

changed does generally increase the plan length as the most efficient way to solve problems with interdependent goals is often to achieve many goals in combination with each other. Solving goals other than the original fixed goal can be done, in domains where operators have inverses, by using the macro-actions to achieve the fixed goal from the given problem state and combining this with the inverse of the result of planning from the required goal state to the fixed goal state. This has a cost, however, in roughly doubling the length of the solution plan required.

Macro-action generation is the most expensive part of this process with most of the problem solving effort being invested in it. The generation algorithm uses iterative deepening search starting from the goal state in order to obtain macro-actions. Search begins backwards from the goal state and when an operator sequence is found that satisfies the criteria to be placed in the table it is inserted. The termination criterion for this search is, however, somewhat more complex. In some problems it is not possible to find a macro-action for every cell in the table, indeed the 8 puzzle is one of these problems as the final two tiles must be in their goal state when all other tiles may no longer be moved. Termination can, therefore, either be determined by a fixed resource cut off; or avoided by interleaving the problem solving and macro-action generation phases requesting new macro-actions as they are required. The approach was successfully applied to a number of puzzle domains including the 8 puzzle, Think-a-Dot game and the towers of Hanoi problem. For larger problems, such as Rubik's Cube and the 15 puzzle, a more sophisticated macro-action generation technique involving a bidirectional search was used.

2.2.7 FM

The FM system [52, 51] learns sets of macro-actions based on past experience of solving problems. Macro-actions are generated from solutions to previous problems. They are consist of the actions in the plan between points at which there were distinct choices of which operator to select. Macro-actions are then generalised to parameterise their preconditions and effects, making them more reusable. The macro-actions are compiled into primitive operators and then used in search.

In order to address the increased branching factor in the search space the concept of 'chunks' was introduced. Chunks are used to guide search in selecting which of a large collection of macro-actions to use, in particular which parameters to instantiate a macro-action with in order to achieve the desired goal.

Two different types of chunks were created, initially the chunks were called

'B' chunks and later evolved to become 'C' chunks. Chunk creation is based on the weakest preconditions of a macro-action and the preconditions of the final action in the macro-action. The chunks consist of a conjunction of predicates, if there is a consistent binding for the chunk then the macro-action instantiation to be selected is one that gives rise to this consistent binding.

Later work by McCluskey and Porteous [53] investigating the engineering of domain models for planning considered macro-actions as a means to improve search performance. When a domain is created search guidance can be learnt in order to allow planners using the domain to plan more efficiently. Macro-actions are one form of such search guidance. Their macro-action generation is based around the ideas used by Korf [43]. Abstract plans are generated to achieve tasks for classes of objects. These plans then become macro-actions for use in solving problems in the domain in question.

The authors also identify five important factors in determining whether a macro-action is useful these are: the likelihood that the macro-action is useful; the search effort saved in using it; the cost of generating the macro-action; the degradation in solution quality through using the macro-action; and the cost of determining applicability of the macro-action during search.

2.2.8 Case-Based Planning

Case based planning [68] allows the planner to learn from previous experience by maintaining a library of *cases*. Cases consist of solution plans for problems with additional information about the success or failure of various decisions in search. The library of cases is built as planning problems in the given domain are solved. The libraries can become large so querying strategies are required to discover which of the cases is the most applicable in a given situation. When appropriate cases are identified the planner uses them to guide search in solving other problem instances in the same domain.

2.3 Modern Planning Approaches

Following several years of popularity of planning as plan-space search Graphplan [4] presented a new structure for searching to find plans. This greatly increased the speed with which plans could be generated. Around the same time, forward heuristic search was becoming a popular way to solve planning problems. The Graphplan and forward search planning strategies were later combined to form

a new generation of forward heuristic search planners; with the planning graph structure being modified to provide a useful relaxation heuristic for planning.

2.3.1 Graphplan

The planning graph used in Graphplan was first described by Blum and Furst [4]. The planning graph structure consists of separate, alternating layers: one containing facts; the next containing actions. The first layer of the planning graph consists of the facts that are found in the initial state, I, of the planning problem. The second layer consists of all actions of which the preconditions are satisfied in the initial state. It may, of course, be the case that not all of these actions can be applied at once, for example when one action deletes a precondition of another. In these cases there is a need to mark actions as mutex if they cannot be applied at the same time, these mutex markings are stored in the graph[3]. Special actions known as 'no-ops' that represent applying no action, are added to each action layer to allow the predicates found in the previous fact layer to be carried forward to the next.

Following the first action layer is another fact layer which contains all of the facts achieved by the actions in the preceding action layer. All facts whose achieving actions are marked as mutex are themselves marked as mutex. The construction of the graph continues from here, alternating fact and action layers, until all of the goals appear and are not marked as pairwise mutex. At this point search begins, backwards from the final layer, to attempt to find a valid plan. If no plan is found searching from a given layer the graph is expanded by two more layers—one more action layer and one more fact layer—and search begins again from the newly created fact layer.

During the search to find a plan from a given layer any *goal sets* (sets of predicates to be achieved in an action layer during search) that could not be achieved are memoised. This is a form of online learning: it allows the planner to avoid unnecessary search if the graph is later expanded by one layer. If a set of goals could not be achieved between the initial state and layer n when searching in a graph of size m then it cannot be achieved between the same two layers when searching from layer $m + 1$. This means that if graphplan when searching from a given layer finds that the goals it needs to achieve at layer l are already memoised as an unachievable goal set, there is no need to continue

[3]Only pairwise mutex relations are considered as capturing n-way mutexes would result in graph construction becoming too computationally expensive. The result of this is that backtracking search must be done in order to solve the planning problem.

attempting to find achievers for this goal set as it has already been shown to be impossible, therefore allowing search to continue in other, more useful, directions. Memoised goal sets have another benefit: it is from these that Graphplan derives the termination condition. Search terminates and Graphplan reports that no solution exists if the number of memoised goal sets is not decreasing. This gives the guarantee that Graphplan will terminate if, and only if, no solution exists.

Following graphplan other planners were built based on the planning graph structure. These include STAN [48], looking at more efficient ways to store the planning graph structure and aspects of problem structure that could be exploited in order to solve problems more efficiently, and IPP [42], investigating the use of more complex language constructs used in ADL in the graphplan framework.

Local Search in Planning Graphs: LPG

The planner LPG [26] built on the Graphplan planning paradigm introducing a technique of using local search through a search space consisting of planning graphs. Each state in the search space consists of a planning graph structure with a number of actions selected within it, representing a partial plan. The neighbourhood used for the local search allows the planner to do one of three things: add an action within an existing layer to support an unsupported precondition; insert a layer into the graph; or remove a selected action from the partial plan. The search continues using the relaxed planning graph heuristic (making use of the solution to a relaxed version of the problem, ignoring delete effects) to select from the neighbourhood. When a local minimum is reached Lagrange multipliers are used to update the costs of difficult to resolve conflicts. Search then begins again from the initial state with the updated weightings causing the planner to explore a different part of the search space. One particularly interesting feature of LPG is that it can perform optimisation of plans, with respect to some metric, by continuing search once a plan has been found.

2.3.2 Early Attempts at Forward Heuristic Search

Forward Heuristic Search through state-space is a popular way to solve planning problems. The planner begins from the initial state, I, and generates its successors, the successor states are then themselves visited in some order determined by the heuristic and the type of search used. The successors of a state S are all states that can be reached from S by the application of a single action in the domain. More formally, for a planning problem $< A, I, G >$ the successors, S_{succ},

of a state, S, are defined by:

$$S_{succ} = \{\forall S' : \exists a \in A : pre\_a \subseteq S \wedge S' = (S \setminus del\_a) \cup add\_a\}$$

Where pre_a, add_a and del_a are the sets of preconditions, positive effects and negative effects of the action a.

Some search strategies use a heuristic as the guide to select the next state for expansion; others combine this idea with further techniques to prune states from the search space.

UNPOP

The planner UNPOP [55] broke from the partial order planning paradigm using forward chaining search to find solution plans. It used regression graphs in order to guide search towards the goal. Regression graphs are computed by selecting all actions that achieve any remaining unachieved goal, and then recursing to select actions that achieve the preconditions of the selected actions until all preconditions of the required action sequence are found in the current state[4]. Search proceeds forward from the initial state, considering only those actions that are in the regression graph, of which the preconditions are satisfied in the current state. The heuristic used for a given state is the number of actions in the plan to reach that state from the current state, plus the estimated distance to the goal state from that state taken from the regression graph. The estimate of goal state distance is the shortest path through the regression graph of which a given action to be applied is the first step.

HSP

HSP[5] (Heuristic Search Planner) was used to investigate planning as search through a space of world states. Many different versions of HSP exist, investigating different search strategies and heuristics. Two possible heuristics were investigated, h_{add} and h_{max}. The calculation of these heuristics is done using a relaxed problem where the delete lists of the actions in the domain are ignored.

[4]A depth cut-off is used so that regression action sequences resulting in preconditions that can never be achieved in the current state do not cause infinite recursion.

The cost, $h(g)$, of achieving a single goal, g, in a state, S is given by:

$$h(g) = \begin{cases} 0 & \text{if } g \in S \\ \sum_{p \in pre_A} h(p) & \text{if } \exists A : g \in add_A \\ \infty & \text{otherwise} \end{cases} \tag{2.1}$$

The two heuristics are defined as follows for each state in a problem with goal set G:

$$h_{add}(G) = \sum_{g \in G} h(g) \tag{2.2}$$

$$h_{max}(G) = \max_{g \in G} h(g) \tag{2.3}$$

The h_{add} heuristic assumes that no interaction between goals occurs when solving the problem, the value it gives is the sum of the costs of achieving all the goals independently. It is, however, often the case that helpful interactions occur in the solutions to planning problems, where solving one goal takes the search process considerably closer to solving another. The h_{max} heuristic is admissible: its value is defined to be the cost of achieving the single goal with the highest cost estimate. Despite the fact that h_{max} is an admissible heuristic h_{add} proved to be the most successful when used in search as it is a more realistic estimate of the cost of solving the problem in most situations.

HSP version 1 performed hill-climbing search forward from the initial state using the h_{add} heuristic. Upon expansion of each state, S, all the possible successors of S are generated and the heuristic values are calculated for each of these states. The successor state with the smallest value of h_{add} is chosen as the next state to be expanded. To avoid the planner becoming stuck in large plateaux where the value of h_{add} does not improve, search restarts from the initial state after expanding a fixed number of plateau nodes. Search continues until a state satisfying the goal conditions is reached. A further version of HSP, HSP 2.0, was created to address the problem that the hill-climbing search approach was not complete. The second version uses a WA* algorithm[44, 61] to search; nodes are expanded in order according to their heuristic value, again using h_{add} but this time also factoring in the cost of achieving the given state from the initial state. The heuristic value, $h(s)$, of a state s in this strategy is equal to $f(s) + W h_{add}(s)$, where $f(s)$ is the cost of reaching s from the initial state (i.e. the number of actions applied so far to reach that state) and W is a constant. HSP 2.0 was able to solve problems more quickly than HSP.

Another version of HSP, HSPr, was also made. This was intended to address the issue that much of the search time was taken up in doing heuristic calculations. HSPr performed regression search backwards from the goal so that heuristic computations could be re-used. This approach proved to be less successful than the forward-chaining search attempts due to the increased search space size caused by the generation of states that cannot be reached from the initial state and the difficulty in defining which state to search backwards from. There are many states in which the goal condition holds, and it is not clear which of these to begin from.

2.3.3 The Development of Planning as Forward Heuristic Search

Modern planners using a forward-chaining search strategy benefit from the heritage of both graphplan-based and forward-search planners. The development of a new heuristic based on the planning graph structure revolutionised planning as forward chaining search providing much better guidance for these search techniques than ever before.

2.3.4 The Relaxed Planning Graph Heuristic

In this section the relaxed planning graph heuristic is discussed. The heuristic is based on the planning graph as discussed in section 2.3.1. The heuristic value calculation is done frequently during search usually for every state encountered, therefore in order to use the technique to form a heuristic the graph construction and extraction of a plan must be relaxed to allow the computation to be done efficiently. The search process of graphplan is equivalent to solving a planning problem, which is P-SPACE hard [19]; this clearly cannot be done at every state. To overcome this problem in the Relaxed Planning Graph the delete effects of the actions are ignored when constructing the graph and searching for a solution plan.

To construct the Relaxed Planning Graph from a state S the first layer is initialised to contain all predicates that are true in S. All actions in the domain with precondition sets that are subsumed by the initial layer are then added to the first action layer (including no-ops to allow the predicates in the first layer to be carried forward to the next layer). The second fact layer contains all the add (positive) effects of the actions in the first action layer, delete effects are ignored so no mutex relations exist. The graph is expanded forwards until all of the goal

conditions appear in a fact layer.

When graph expansion is complete a greedy algorithm is used to generate a *relaxed plan*. The search starts from the goal state and selects the first achiever found for each of the goal predicates. The preconditions of the selected achievers are then considered and an achiever is selected for each of these. The process continues until all the achievers are contained within the initial state. Although finding the optimal relaxed plan is NP-Hard [9]; using this algorithm to find an arbitrary relaxed plan is polynomial in the size of the ground domain.

2.3.5 FF and Enforced Hill-Climbing

The popularity of forward-chaining search was revived by FF [32] after its performance in the Second International Planning Competition. FF, like HSP, uses a forward-chaining search strategy through a search space made up of world-states; however, in FF, the precise nature of the forward-chaining search used is different, as is the heuristic [37]. FF uses the relaxed planning graph heuristic to guide it during enforced hill-climbing (EHC) search, a hill-climbing local-search approach.

The algorithm in Figure 2.1 describes formally the process of enforced hill-climbing. Informally it can be described as follows. The successors of each state, S to be expanded, are generated in an arbitrary order (that in which the actions appear in the domain). When each successor is generated its heuristic value is calculated. If a successor state, S', with a lower heuristic value than S is found, search continues from S' expanding its successors.

If all the successors of a state have been expanded and no state with a strictly better heuristic value has been found the search has arrived at a *plateau* in the local search landscape. Search cannot continue as before, by selecting the first successor with a strictly-better heuristic value, as all of the successor states have a heuristic value that is greater than or equal to that of the state to be expanded. At this point an alternative strategy is required and FF resorts to breadth-first search. Breadth-first search continues until a state with a strictly-better heuristic value is found; when such a state is found the end of the plateau is reached and search continues as before. It is worth noting that since states are generated in an order dictated by the order that the actions appear in the domain description, and enforced hill-climbing is a fastest, not steepest, descent procedure, the performance of the search is sensitive to action ordering.

EHC does not maintain completeness; FF, however maintains this property in search by using a best-first search algorithm to follow the enforced hill-climbing process. If no plan is found via EHC FF resorts to best-first search, using the

Require: I, the initial state from which to begin planning, A, set of instantiated action operators, G the set of propositions required to be true in a goal state

```
1: Openlist ← I
2: while S ←pop head from Openlist do
3:   for each A_i ∈ A • pre_{A_i} ⊆ S do
4:     S' ← S \ del_{A_i} ∪ add_{A_i}
5:     if heuristic value(S') < heuristic value(S) then
6:       S ← S'
7:       clear Openlist
8:       Openlist ← S
9:       if G ⊆ S' then
10:        return Path to S'
11:      end if
12:      break for loop
13:    else
14:      push S' onto tail of openlist
15:    end if
16:  end for
17: end while
```

Figure 2.1: Enforced Hill-Climbing Search Algorithm

same heuristic, in order to find a plan should one exist.

Helpful-Action Pruning

FF uses the relaxed planning graph not only to give a heuristic value for each state; but also to prune action choices at each branching point. Upon the expansion of each state there are many possible successor states to consider. The fewer states the planner has to consider before finding a strictly-better state, the more quickly search can proceed towards the goal. The aim of helpful action pruning is to identify those actions which are more likely to lead to a state with a better heuristic value and prune all the other actions so that the goal can be reached more quickly.

When calculating the heuristic value of states the relaxed plan is extracted from the relaxed planning graph but only its length is used. For helpful action pruning the relaxed plan itself is used. The assumption that the relaxed plan is a reasonable approximation to the final plan suggests that the actions in the first layer of the relaxed plan are good potential choices for the next action to apply and further, that if an action does not appear in this first layer it is less likely to lead to a state closer to the goal. As such FF maintains only the ground

actions that appear in the first step of the relaxed plan or actions which achieve an effect achieved by such actions. This pruning is done during EHC search and breadth-first search on plateaux; but not during best-first search as it would affect completeness. This is particularly useful on plateaux, when it is often the case that all the successor nodes of a given state have to be expanded as no strictly better state can be found. With reference to figure 2.1 the helpful-action pruning is applied at line 3 where the condition that the action is applicable in a state is replaced with the condition that the action is helpful (helpful actions are necessarily applicable as the first layer of the relaxed planning graph consists of all applicable actions in the current state).

Search Landscape Topology Under the $h+$ Heuristic

In 2001 Hoffmann performed both an empirical [33] and a theoretical [34] analysis of the search landscape under the $h+$ heuristic. The $h+$ heuristic is the optimal relaxed plan to solve the relaxed planning problem. As the relaxed planning graph heuristic used in practice in FF is an approximation to the optimal relaxed plan, the conclusions from this analysis have great relevance to the search performance under the FF's heuristic. The topology of the search landscape can be used to deduce which problems a planner using this heuristic will solve efficiently and which are likely to prove more difficult. The full investigation [36] considered 30 domains including all of the domains used in planning competitions up to the date of publication. In order to improve the performance of an already strong search algorithm it is important to note where the current weaknesses lie.

The study investigated several properties of standard benchmark domains and demonstrated the features of those domains that were allowing forward-heuristic search to find solutions in a short length of time. More difficult problems were found to be characterised by dead ends and the presence of local minima and plateaux in the search space.

A noteable feature observed in some domains is that the maximal exit distance is less than a given constant (and is often the same) for all problem sizes. The maximal exit distance is the number of actions required to escape from a *local minimum* (states where all neighbouring states have a greater heuristic value) or a *bench* (states for which all neighbouring states have equal heuristic value). Such regions are the regions in the search space at plateaux, where the heuristic cannot find a strictly-better state. In such domains plateau-escaping sequences will be of similar, or equal length. The search spaces of a number of benchmark problems were shown to not contain any local-minima although they do still

contain plateaux in the form of benches.

The search space can be defined as either undirected, meaning that no dead ends occur, or directed. Directed search spaces, those containing dead ends, are split into three distinct categories: harmless, recognised and unrecognised. Search spaces are *harmless* if they are directed but contain no dead ends; *recognised* if all dead ends are trivially shown to be dead ends by an infinite heuristic value; and *unrecognised* if they contain at least one dead end state where the heuristic is not infinite. The empirical analysis showed that problems leading to search spaces containing unrecognised dead ends (such as FreeCell and Mystery) were the most difficult. Recognised search spaces were the next most difficult category, followed by harmless search spaces and finally undirected search spaces. The dead-end properties of the search space will be important in considering the use of macro-actions: care must be taken as macro-actions could potentially lead search into an unrecoverable dead end.

2.3.6 YAHSP

The YAHSP planner [69, 70] is a forward-heuristic search planner that uses best-first search along with the heuristic and pruning strategies of FF. The premise upon which YAHSP is built is the notion of trust in the correctness of the heuristic. Planners before FF used the heuristic only for the numerical value it produced to evaluate which successor states to expand; the helpful-action pruning in FF extended the use of the relaxed plan extracted by using it to prune actions from the search space. Motivated by the observation that the relaxed plan is often a good guide to the solution to the problem, YAHSP extends the use of the relaxed plan further to generate a successor state by following some of the actions in the relaxed plan.

The search algorithm used in YAHSP is complete, unlike the enforced hill-climbing approach[5]; YAHSP still uses the notion of helpful actions but simply uses a best first search approach where the successors generated using helpful actions are always considered before those generated by the other actions, termed *rescue actions*, which remain in the search queue to preserve completeness. YAHSP uses a slightly modified version of the relaxed planning graph, again inspired by the notion of trust in the heuristic, the relaxed planning graph is built using only *preferred actions*. Preferred actions are actions that do not delete facts that are not present in the initial state but are present in the goal state. The application

[5]FF still remains a complete planner as it resorts to a complete best-first search if no plan is found via enforced hill-climbing.

of actions which do not fall into this category would imply that the heuristic had suggested achieving a subgoal only to destroy it again. If the goals do not appear in this new relaxed planning graph, the original planning graph constructed using all of the actions in the domain, as in FF, is used.

The notion of using a sequence of actions extracted from the relaxed plan is similar to that of using a macro-action to guide the search: these lookahead plans can be thought of as macro-actions generated from the heuristic. The lookahead algorithm begins with the relaxed plan for the state S and creates a new state S' and a valid sequence of actions P that transforms the state S into the state S'. All actions in the relaxed plan that are applicable in the state S are applied to initialise S' and P is initialised to contain those actions. Then, for the first action a that cannot be applied, the algorithm searches for other actions, not in the relaxed plan, which are applicable in the current S' and add a precondition of a. The best of these actions (the action with the minimal sum of the levels at which its preconditions arise) is applied to S' and added to P. The algorithm then tries to apply the actions of the relaxed plan that could not be applied in the last iteration and the process repeats until it is not possible to add any more actions to the relaxed plan that fit the criteria above. When the algorithm terminates S' is the new state that is added to the search queue, with P being the sequence of actions to achieve S' from the current state, S.

2.3.7 Fast and Diagonally Downward

The RPG heuristic is just one example of a relaxation heuristic; in this case computed by ignoring the delete lists of actions. The heuristic used in Fast Downward [28] introduced a different relaxation approach. The relaxation used in downward is based around the SAS+ formalism [2], an alternative method for expressing planning problems. In the STRIPS formalism each state consists of a number of predicates which describe the world state; states in the SAS+ formalism consist of a number of variables which can take multiple values from a finite domain. The preconditions and effects of actions are modelled as partial variable assignments. Using this formalism planning problems can be expressed: the initial and goal states are collections of variables with associated domains and the preconditions and effects of operators consist of partial variable assignments.

Translation between the STRIPS formalism and the SAS+ formalism can be done simply. It can be achieved by creating a variable for each of the predicates in the planning domain which can take the value either true or false and changing the preconditions and effects of actions to assign the value true or false to the

variables they represent. The automated translation done in this way is correct, however, in many domains a more useful representation could be generated by introducing variables with non-binary domains. For example, in a logistics domain the position of a truck could be represented by a single variable with a domain that allows it to take the value of any location in the map.

The downward planner [29] has two versions. The first of which simply uses a best first search using the causal graph heuristic. The second version, diagonally downward, uses both the causal graph and RPG heuristics taking actions alternately from two open lists: one corresponding to each heuristic.

2.3.8 The Causal Graph Heuristic

A *causal graph* is created from the SAS+ domain in order to perform the computation of the causal graph (CG) heuristic. The nodes in the graph are the variables in the domain. Directed edges are created between pairs of nodes where a variable represented by one node appears in the precondition of an action that changes the value of another. Further edges are added between variables of which the values are changed by the same action. The edges in the graph represent a causal dependency between the two variables. Cycles are removed from the causal graph in order to relax the problem to allow a heuristic value to be calculated from it. Edges are removed according to the principle that the most important part of problem structure to be maintained is the structure related to the most-constrained variables. To this end, the edges broken in order to remove cycles are those into nodes which have the fewest outgoing edges; that is, those variables which have the lowest number of variables dependent upon them.

Computation of the heuristic from the causal graph is done by a recursive procedure starting from the nodes which have no edges emanating from them (i.e. no variables dependent upon them). The algorithm computes the shortest path through the graph that can be used to change the value of each variable in the graph, with dependencies, to the value it must take in the goal state (if specified).

Preferred Operators in Downward

The use of preferred operators in downward was inspired by the helpful actions used in FF [32] (see section 2.3.5). The idea is to use the structure generated in calculating the heuristic value to suggest certain actions that may be useful in achieving goals. Under the RPG heuristic these are all actions that achieve any

effects achieved by an action in the first layer of the relaxed plan; under the CG heuristic they are actions that are represented by *helpful transitions*.

Recall that each arc in the causal graph represents one of two things: either a variable is a precondition of an action achieving another variable, or two variables are mutually affected by the same action. When computing the heuristic value a number of transitions in the graph are considered to find the shortest path to allow the variable to reach the value it takes in the goal state. The helpful transitions are found by considering the first transition made on the path generated to the goal of each operator; i.e. the transition that is taken directly from the value of the variable in its current state. If this transition corresponds to an operator that is applicable in the current state then the operator in question is considered as a helpful. If the operator is not applicable in the current state, due to some other unsatisfied precondition on another variable, the algorithm recurses to find operators that are helpful in achieving the unsatisfied preconditions. The helpful-transitions are calculated for every variable whose value is specified in the goal state. It is possible that this algorithm will not find any operators that correspond to the helpful transitions in the graph, in which case the helpful actions generated by the RPG heuristic can be used. Fast downward uses the CG helpful transitions falling back on the RPG helpful actions if necessary; diagonally downward uses both the RPG helpful actions and the CG helpful transitions in a hybrid search approach.

2.4 Modern Approaches to the use of Macro-Actions and Learning From Previous Planning Experience

Recent macro-action generation techniques often capitalise on the recent progress made in the field of machine learning. Offline learning techniques are used to find and filter macro-actions, with a small collection of macro-actions being added to the domain for use in search. *Offline* learning techniques are those in which learning occurs before the problem solving phase, consuming additional time beyond that required to solve the problem. In contrast *online* learning techniques learn only during the problem solving phase itself, no additional learning time is required for the system to learn: it is all included in the time taken to solve the problem.

2.4.1 Macro-FF

Macro-FF [8] is a planner that uses an offline approach to learning macro-actions. The planner requires a number of small problem instances for each domain it is used to solve. Analysis is performed on the smaller problems in order to derive potentially useful macro-actions. Two versions of macro-FF exist, one using component abstraction to find macro-actions; the other extracting macro-actions from solution plans. Both approaches use a filtering technique making use of a training process to find a small number of macro-actions to use when planning.

The training process for the first approach [6] involves creating abstract components. These are sets of objects in the domain that are linked together by static predicates. Linking of objects in this way indicates that they may well be part of a larger object which may therefore allow abstraction. For example, in the rovers domain, from IPC 3 [49], each rover has a number of tools associated with it. This is modelled in the domain by a predicate linking each piece of equipment to a rover, for example:

(on_board camera1 rover1)
(store_of store1 rover1)

Further predicates may also pertain to to the components of the rover, such as:

(supports camera1 mode1)

Component abstraction can be used to detect that predicates describe one larger component of the domain built using these predicates. The process begins by selecting a type at random, for example the store type. For every object of type store in the problem instance a separate component is made. Once the components have been initialised static predicates are taken from the domain until one which refers to an object already in a component. Consider the component that contains store1 in this example, call this component1, the predicate (store_of store1 rover1) will be found in the domain. The object rover1 will be added to component1 as it is connected to a store1 which is already in component1. Expansion of all components continues in the same way repeating search over all predicates in the domain for every object that is added to a component. This algorithm is slightly modified to ensure that components do not become too large: if a predicate connects two or more existing disjoint components then it is not

used to expand either component, it is simply ignored.

The static components extracted from the domain are used as a basis for the formation of macro-actions. A process of forward search is used to build an initial large population of macro-actions. Beginning with an empty macro-action an action is added to the end of the macro-action at each expansion in the search space. Expansion of macro-actions is not done entirely arbitrarily, only actions producing macro-actions that adhere to a certain set of rules are permitted:

- New actions must not have a precondition that is deleted by the current macro-action (to help to maintain soundness of the actions).
- Macro-actions must not contain cycles: otherwise a shorter acyclic macro-action could be found.
- Any action in the macro-action (except the first) must have as a precondition one of the effects of the action that precedes it. This is an attempt to ensure that the macro-action represents a linked and interdependent sequence of actions in the real world rather than just a chain of independent actions.

The connected components analysis is also used to limit the scope of the search: macro-actions are only permitted to modify objects from one static component. Adding an action to the macro-action is only allowed if it does not cause the macro-action to modify more than one component. The algorithm has limits on macro-action length and number of preconditions.

The search process generates many macro-actions; it is, however, not desirable to add many macro-actions to the domain due to the increased branching factor encountered. A filtering algorithm is therefore used to select only the two most promising macro-actions for inclusion in an enhanced domain. The process involves solving a number of small training problems; each of these is solved, by FF, using all of the macro-actions. Macro-actions useful in small problems are assumed also to be potentially useful in larger problems and are therefore weighted based on the number of times they are found in solution plans. The two macro-actions with the best weighting at the end of the process are selected to be added to the domain. The enhanced domain contains compiled macro-actions and can therefore be given to an unmodified planner for use in solving problems.

Macro-FF implements another approach to generating macro-actions [7]; this second approach works independently of the first, the two are not used in combination. Instead of performing analysis on the small problem files; macro-actions

are extracted from solution plans generated by solving the small training instances. The number of macro-actions extracted is kept small by analysis of the solution plan to find connected action sequences. The macro-actions generated are then pre-filtered according to the conditions listed above (excluding the connected components analysis and length restrictions). An additional static filtering condition is added to remove macro-actions that overlap in the solution plan. For example if the macro-action A-B-C-A was found and plans frequently contain the sequence A-B-C-A-B-C it would be preferable to keep the action sequence A-B-C, so that the macro-action could repeated.

A dynamic filtering technique allows for further pruning of the list of macro-actions. This technique is invoked once after a fixed number of runs of the planner using the macro-actions to solve problems. Filtering is based on the *efficiency rating* of the macro-action which is calculated based on the number of times the macro-action is instantiated and the number of times it is used in a solution plan. Macro-actions with low efficiency ratings at the time the dynamic filter is run are pruned from the list.

When the macro-actions are used in the planning process they are considered before other actions, with the motivation that they may allow a jump through the search space without the need to consider all of the unit actions first. To restrict the number of possible groundings a concept known as *helpful macro-actions* was developed, this is similar to FF's helpful actions. For a macro-action instantiation to be considered helpful, and therefore be considered during planning, at least k of its steps must be found in the relaxed plan.

2.4.2 Evolving Macro-Actions

Recently work has been done on learning macro-actions using a genetic algorithm approach [59]. The approach, along with that used by Macro-FF, uses an off-line learning algorithm to generate several candidate macro-actions and filter them to select the most promising two macro-actions to add to the domain. Both approaches are domain and planner independent. The difference between the genetic algorithm approach and that of Macro-FF is the filtering algorithm used. Both approaches acquire an initial population of macro-actions by searching through solution plans to lift random macro-actions but a genetic algorithm is used to prune the macro-actions in the second approach.

A genetic algorithm [45] begins with a seeded population and then performs cross-over and mutation operations on certain population members (selected according to a given fitness function) to generate a new population from the old

population. This continues recursively on newly generated populations until either a certain level of fitness has been achieved or a certain length of time has elapsed. In this case, an initial population of macro-actions is generated by randomly lifting macro-actions of different lengths from plans generated by solving small problem instances in a given domain. The cross-over and mutation operators are then applied to these actions.

The cross-over operator used is simply to take the first n actions from one macro-action and add the last m actions from another after these to create a new macro-action; the remaining actions from the first macro-action are then added to the end of the remaining actions from the second to create another new macro-action. Motivated by a desire to keep short macro-actions only the shorter of the two resulting actions is considered as a member of the next population. Three different types of mutation are used: addition, adding an action at a random place in the existing macro-action; deletion, deleting a random action from the macro-action; and alteration, replacing a random action in the macro-action with another action from the domain. A number of additional conditions, similar to those used by Macro-FF, are used to prune inconsistent macro-actions and those which are considered too large.

The fitness value of each macro-action in the new population must be calculated. Evaluation is done by running the planner on a number of assessment problems. Assessment problems are generated problems which are harder to solve than the small problems (the solutions of which were used to lift the macro-actions that seeded the problem) but are still solveable in a relatively short time (between 1 and 10 seconds). Each assessment problem is solved once with the original domain and again individually with a domain containing each macro-action. The fitness value for a given macro-action is calculated based on the time taken to solve the problem with that macro-action in the domain and the probability that the problem will be successfully solved within the time limit. When a chosen number of generations has been evaluated the two macro-actions with the highest fitness value are added to the domain and further problems are solved using this enhanced domain.

2.4.3 Planners Using Action Reordering

Action Reordering Based on Almost Symmetry

Symmetry has been used in systems, such as Symmetric STAN [24], to prune the search space allowing action choices that are not usefully distinguishable to be

pruned from the search space. A development of the work on symmetry [23] was to use symmetries positively, to suggest actions, rather than negatively, to prune actions, through action reordering. The approach taken was to order the helpful actions, as generated by FF, to consider first those actions that were almost symmetric with actions applied at some previous time point in the plan. Almost symmetric actions are actions that are symmetric, in the traditional sense, in an abstracted version of the domain. The abstraction allows more actions to be considered symmetric than in the conventional definition of symmetry by ignoring some of the bindings in the predicates describing an object. For example, in the blocksworld domain the predicate (on x y) may describe the property of some block x; while the predicate (on p q) describes the property of a similar block p. The actions cannot be pruned on this basis since the blocks are not symmetrical (they are not on the same block) but they are almost symmetrical in the sense that their situation is similar and the same actions may well be useful. Since the same actions that have been used to transform x to its goal state may well be useful in transforming p to its goal state such actions are considered first.

CRIKEY

CRIKEY [27] is a temporal planner that creates its plans by first generating a sequential plan using FF and then using a scheduler to parallelise and schedule the actions in the plan. CRIKEY uses action reordering when planning in an FF style to maximise the potential for parallelisation of the finished plan. Actions are ordered dynamically at each stage in planning according to the last action in the plan. The ordering of the helpful actions (the only actions to be considered) is such that all actions that are not mutex with the action currently at the end of the plan are considered first; followed by all of the actions that are not parallelisable with it. The result is that if an action that can be performed at the same time as the previous action leads to a strictly better state it will be used in favour of an action that cannot be parallelised. The action reordering allows for greater parallelisation of the final plan and improved makespans of the plans generated by CRIKEY.

2.4.4 Learning Control Rules for Planning

Planners traditionally rely on a generic heuristic and no domain specific knowledge (other than the specification of the initial and goal states and the actions themselves). Some systems, however, are able to load user-specified control rules

that encode additional search guidance. Systems able to make use of this extra knowledge, include TLPlan [1], TALPlanner [46] and SHOP[6] [57]. The control rules used in these systems had to be hand coded, the process of devising and encoding these control rules was very labour intensive.

Just as macro-actions can be learnt in planning, it is possible to learn control rules for planning. Early approaches to doing this were made in the HAMLET system [10]; the control rules generated by HAMLET were used in the Prodigy system. The learning process takes place in three stages: first control rules are learnt from the path taken during search to find a solution; secondly those rules are generalised to allow them to be more widely applicable; finally, the rules are refined, by learning negative examples from the search tree, to ensure that the rules are only used when it is correct to do so. Martin and Geffner [50] developed a system that allowed sophisticated control rules to be generated through allowing new concepts to be learnt using description logics. The language allowed the planner to learn the concept of a 'well placed' block in the blocksworld domain (i.e. one that has reached its final goal destination); many previous systems learning control rules were given this concept by human operators before learning began.

A recent approach, taken in the L2plan system [47], is to use a genetic algorithm to learn control rules. The rules consist of three parts: a condition stating the facts that are true in states where the rule should be applied, a goal condition, linking the objects in the condition to their goal configuration and an action offering guidance to the planner about the next action to apply if the condition holds. The rules are initialised randomly, with arbitrary conditions and action suggestions. The rules are combined together to form policies and are then subjected to evaluation, mutation and crossover forming a genetic algorithm approach. Once a given policy of rules reaches a certain level of fitness on the set of training problems the process terminates with a policy of rules which the planner will then use during planning. The control rules generated were shown, not only to improve performance over not using rules, but also to outperform a hand coded policy written by the authors of the paper. A dynamic action reordering strategy can be thought of as a type of control rule. Control rules condition on the predicates in the current and goal states to suggest a next action; action reordering strategies can condition on the same thing [23] or the previous action [27].

The learning of control rules has been linked to the use of macro-actions

[6]SHOP uses task hierarchies, which are control knowledge provided in a slightly different format.

[17]; this has been done in a planner named IPSS. Macro-actions are lifted from solution plans, sequences selected are those that occur the most frequently in solutions. The control rule learning mechanism is then applied with the macro-actions in addition to the original domain. The rules learnt can then incorporate macro-actions.

2.5 Chapter Summary

This chapter has given an overview of previous work that is related to the contributions presented in this thesis. An introduction to the field of domain-independent planning has been given detailing the development of planning systems over recent decades and explaining the emerging search paradigms. Forward-chaining planning was observed to be one of the most successful search paradigms of recent times, this is the basis on which the work in this thesis builds.

Planning remains a hard problem, despite great advances in producing generic heuristics to solve problems. It can be seen that learning has allowed planning performance to be further enhanced bringing it closer to the ideal of being able to tackle difficult real-world problems. The learning of macro-actions in planning is an area that has received a great deal of research interest. Recent attempts at learning macro-actions have used offline learning steps: requiring that the planner solves training problems before beginning to solve the problems of interest. In some cases, however, such small problems may not exist; may have to be specifically made by humans; or may not represent the same structure as large problems. There is, therefore, scope for the production of an online macro-action learning technique in order to improve planner performance without the need for offline training.

Chapter 3

Marvin - a Heuristic Search Planner

3.1 Introduction To Marvin

This chapter introduces the first contribution of the thesis: a technique for the online learning of macro-actions implemented in the planner Marvin [12, 14]. *Plateau-escaping macro-actions* are used to guide search in the appropriate direction when the heuristic is unable to do so. Marvin's search strategy is based on enforced hill-climbing (EHC) as used in FF (see section 2.3.5); however, it adds a number of features to enhance the basic search strategy. These modifications are discussed here to allow comprehension of the framework in which the macro-action learning techniques presented in this thesis are implemented and evaluated. The work presented in the subsequent chapters builds on the features of Marvin and it is used in chapter 5 as a basis for the evaluation of the macro-action learning techniques.

3.2 General Search Behaviour

The search behaviour of Marvin extends the enforced hill-climbing algorithm described in section 2.3.5 and shown in figure 2.1. A variant on this algorithm is used making use of of best-first, rather than breadth-first search on plateaux. That is, on line 14 the S' is inserted into the search queue according to its heuristic value, being placed before all states of higher heuristic value. The heuristic used for search is the relaxed planning graph heuristic, as in FF. This uses the length of the solution to a relaxed version of the problem, ignoring the negative effects of actions, to guide search (the process of heuristic computation is described in

detail in section 2.3.4). The heuristic is not admissible: for the relaxed plan length to be admissible the relaxed plan would have to be guaranteed to be an optimal solution to the relaxed problem; extracting such an optimal solution is NP-hard, so this is infeasible in heuristic computation. Instead a greedy approach is taken to extract *a* solution to the relaxed problem in polynomial time.

Marvin is also extended to make use of more advanced language features, the input language is PDDL 2.1 (excluding numeric or temporal effects) allowing the ADL language extensions [62]. The basic STRIPS language allows actions with preconditions and effects that are conjunctions of literals. The extension to handle ADL allows the use of disjunction in precondition formulæ, the use of existential quantifiers ($\forall$ and $\exists$) and the use of conditional effects. The output from Marvin is a time-stamped sequence of actions to achieve a specified goal; the actions used are taken from those specified in the input.

3.3 Overview of the Modifications Made to the Search Strategy in Marvin

In this section the modifications to the basic search algorithm are described in detail.. The modifications made include allowing actions to be applied concurrently; pruning symmetrical choices in the search space; native handling of ADL [62] and derived predicates; using a modified form of best-first search when enforced hill-climbing fails; and using best-first, instead of breadth-first, search to escape plateaux. The modifications relevant to the generation and use of macro-actions are discussed in this section; details of the other modifications made can be found in [14].

3.3.1 Modifications to Support Concurrency

The EHC search process in Marvin is modified to support concurrency by including additional nodes as potential successors to a given node in the search space. The search nodes in the forward-chaining search approach used by Marvin consist of a time stamp *T*, the world state *S* and the plan so far *P*.

To plan sequentially the successors of a given node are constructed by adding an action at time *T* to the plan *P* to produce *P'*, applying the action's effects to *S* to produce *S'*, and incrementing T by one time unit. As the effects of the action are applied immediately to *S* to generate *S'* there is no opportunity for concurrency. To provide a framework in which planning with concurrency is

possible, the states are modified so that, instead of immediately being modified to reflect the effects of the action, they note the action's effects but do not apply them until time has been advanced.

Each state is expanded to hold an event queue, an approach similar to that used by Sapa [16]. Each entry in the queue is time-stamped and stores four sets: predicates to add when time is advanced, predicates to delete when time is advanced, predicates which are to be held as invariants and predicates to be held as negative invariants. To add a single-step action to a plan the add effects, delete effects, preconditions, and negative preconditions of the action are added to the respective sets in the entry at the head of the queue; to add a macro-action the effects and preconditions of all its component actions are added to the appropriate entries in the queue (reflecting the time at which they occur as part of the macro-action) and the preconditions of future actions are propagated backwards through the queue as invariants to ensure that they are not deleted by a prior action.

When time is advanced in a given state the entry at the head of the event queue is removed, and used to make the appropriate modifications to state.

An action can be applied to a given state, without advancing its time, if the following conditions hold:

action's preconditions $\cap$ queued delete effects $= \emptyset$
action's preconditions $\cap$ queued negative invariants $= \emptyset$
action's negative preconditions $\cap$ queued delete effects $= \emptyset$
action's negative preconditions $\cap$ queued add effects $= \emptyset$
action's negative preconditions $\cap$ queued invariants $= \emptyset$
action's add effects $\cap$ queued delete effects $= \emptyset$
action's add effects $\cap$ queued negative invariants $= \emptyset$
action's delete effects $\cap$ queued delete effects $= \emptyset$
action's delete effects $\cap$ queued add effects $= \emptyset$
action's delete effects $\cap$ queued negative invariants $= \emptyset$
action's delete effects $\cap$ queued invariants $= \emptyset$

These restrictions enforce that the action to be applied cannot be mutex with any action occurring at the time step at which it is to be applied and that it cannot delete any predicates (or add any predicates that are required by a queued action).

To plan with concurrency the successors of a given node are formed by applying actions at all time stamps between T and that of the last entry in the

event queue in S inclusive. As Marvin takes the first strictly better successor when performing EHC, and actions that can be placed in parallel to the existing actions are considered first, plans containing concurrency are produced when it is considered heuristically wise to do so.

In Marvin, concurrency is not considered when using best-first search—either to escape a plateau or as a fall-back when EHC fails—as the increased number of successors produced would otherwise severely increase the branching factor.

3.3.2 ADL Support

Macro-actions are higher-level lifted actions that model activity in the domain in an abstract manner. Reasoning directly with higher-level language constructs can allow a planner to generate more complex macro-actions, and indeed in some cases to generate macro-actions where it was not otherwise possible using Marvin's extraction techniques. Upon the release of the IPC 4 domains [31] it became clear that Marvin could not generate macro-actions effectively from an automatically compiled STRIPS versions of domains originally written in ADL. On examination of such domains it became apparent that many of the actions were already partially ground and had operators that were identical except in name and the grounding of a different constant. These operators are actually compiled from the same original ADL operator; although the planner, of course, does not know this upon being given only the compilation. If a macro-action were to be formed in the ADL domain, using the ADL action that formed all of these operators in the STRIPS domain, it would be much more general and would represent a set of macro-actions that would have to be generated in the STRIPS domain. Furthermore, the ground macro-actions from the STRIPS domain are much less likely to be reusable since they refer to specific entities. If a plateau is escaped for an entity of a given type it is often the case that similar plateaux are encountered for other entities of that type, but if the action is already ground to a specific entity then it cannot be used for other entities.

It can therefore be seen that explicitly handling the ADL domain within the planner, rather than using a preprocessing step to compile the domain, offers potential benefits in terms of macro-action generality: only one action must be added to the ADL domain to represent several in the STRIPS domain and, since the action is not ground, the action is much more likely to be reusable. Marvin handles ADL internally and, for the reasons discussed, does not use a pre-compilation step followed by reasoning with STRIPS actions.

3.4 Macro-Action Generation

Many planning problems exhibit a large amount of symmetry: whilst symmetry in the problem instance itself has been exploited [24, 64]; it is difficult to exploit symmetry in solution plans. The difficulty arises because the final solution plan is not known until the planning process is complete: as such, during the planning process, it is not obvious where the symmetry in the solution plan will be. Symmetry in plans is characterised by recurrent action sequences. Such sequences represent some underlying symmetry between objects in the domain: they often occur because many symmetric objects require the same sequence of actions to transform them from their initial state into the final goal state. If potential recurrent action sequences can be identified during search for a solution plan and encapsulated to form macro-actions a potential reduction in planning time can be made—particularly if these action sequences are difficult to find.

In the process of using EHC to perform forward-chaining heuristic search, guided by the RPG heuristic, plateaux are encountered. Plateaux occur when a local minimum in the search space has been reached and all successor steps require either a sideways move (not changing the current heuristic value) or an uphill move (increasing the current heuristic value). It is these plateaux that are the core difficulty encountered when planning in this manner. It is relatively easy to make progress towards the goal when the heuristic is being informative; however, the exhaustive search performed to escape a plateau consumes a great deal of time.

On inspection of the steps required to escape plateaux in a given domain it is often the case that the same sequence of actions is used to escape many plateaux, but with different parameter bindings. For example, in the depots domain, from the IPC 3, the pattern lift-load is frequently used with different groundings of the actions. In the philosophers problem, part of the Promela domain from IPC 4, a complex action sequence is used many times, once for each pair of philosophers.

As exhaustive search is required to escape from a plateau, construction of plateau-escaping action sequences is computationally expensive. Since plateau-escaping sequences often have similar structure, it is clear that memoising these action sequences for later use—when plateaux are once-again encountered—can potentially reduce planning time. In Marvin the plateau-escaping action sequences are used to construct *plateau-escaping macro-actions*. Once devised, they are made available, as macro-actions, for application on later plateaux. The macro-actions are not compiled into single unit actions for the planner to use, but are instead kept as sequences of actions, thus allowing the planner to reason

about them explicitly to minimise the potential costs of using them.

3.4.1 Extraction of Plateau-Escaping Macro-Actions

Due to the nature of the relaxed problem used to generate the RPG heuristic there are aspects of the original problem that are not captured. As such, when the RPG heuristic is used as an approximation to the optimal (angelically non-deterministic) heuristic to perform EHC, plateaux are often encountered: areas where the RPG heuristic is unable to provide guidance. In these areas, the RPG heuristic value of all successor states is the same as, or worse than, the heuristic value of the current state. The nature of the plateaux encountered, and whether EHC is able to find a path to escape from them, is influenced by the properties of the planning domain [36].

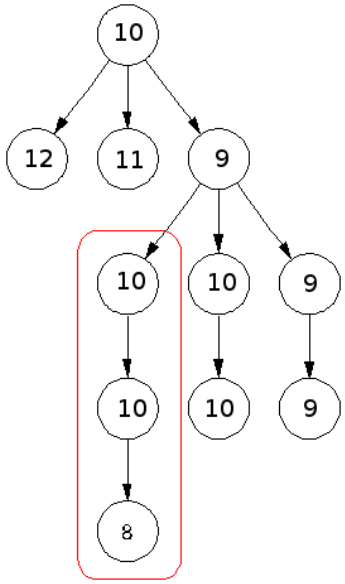


Figure 3.1: Illustration of the Sequence of Actions used to form Plateau-Escaping Macro-Actions

When the start of a plateau is detected—that is, when no successor state with a strictly-better heuristic value can be found—best-first search commences from the current state. During best-first search, each successor state stores the actions that have been applied to reach it since the start of the plateau: when a strictly-better state is eventually found, this list of actions is the plan segment that forms the basis of the plateau-escaping macro-action. An illustrated example is given in figure 3.1. The planner reaches a state at the start of a plateau with heuristic value 9. Exhaustive search begins from this state expanding the successor nodes until a strictly better state is found with a heuristic value of 8. When this state is

reached the actions representing the transitions inside the red rectangle (including the edge emanating from the plateau start node that is partially in this rectangle) are the actions used to form a plateau-escaping macro-action.

In order to make the macro-actions produced as useful and reusable as possible the plan segment is processed before being made into a macro-action. Firstly, any independent threads of execution that exist in the plan are separated to produce macro-actions involving as few entities as possible. As the number of groundings of a macro-action increases with the number of entities in its parameter list, large macro-actions often become problematic: they increase the number of successor nodes to each state, causing the search process to stop making progress. By identifying independent threads, non-interacting entities are separated, splitting the macro-action into two or more actions, reducing the number of ground actions.

Independent threads of execution are identified by inspecting an artificially-created partial-order plan, derived from the total-order plan segment. In a directed-acyclic-graph representation of a partial-order plan segment, independent threads can be identified by extracting independent sub-graphs. The partial-order is inferred as follows:

- for each precondition in the original plan that is satisfied by the effect of an earlier action an ordering constraint (and causal link) is inserted, such that the precondition must come after its achiever;
- for each mutex action pair A and B in the original plan segment, with A scheduled before B, an ordering constraint is added such that B must occur after A.

When matching preconditions to achievers, the latest achiever that occurs before the precondition is used. If there is no earlier action in the plan segment that achieves a given precondition then no dependency is inserted—the precondition forms an *external precondition* of the resulting macro-action.

A forward reachability analysis is computed from each action step in the partial-order plan; two action sequences are independent if they do not share a common reachable action, indicating a common dependency or ordered interaction in the original plan segment. Any action sequences of unit length are discarded as these would create single-step macro-actions which are primitive actions that already exist in the domain.

An Example: The Gripper Domain

A worked example of the macro-action generation techniques will now be presented. The example makes use of the gripper domain, a simple domain in which a robot must transport a number of balls from one room to another. There are three possible actions that the robot can carry out: pick up a ball, using one of its two grippers; move from one room to the other; and drop a ball from one of its two grippers. Clearly although there are only three lifted actions in this domain there are actually several possible action instantiations: for example, if there are 100 balls and 2 grippers then there are 200 possible pickup actions.

Ignoring the delete effects of the 'pickup' action in the Gripper domain creates a problem in which, in a given state, it is possible to pick up many balls using one gripper, so long as the gripper is initially available: the delete effect of the action, marking the gripper as no-longer being available is removed. The relaxed plan in the initial problem state is to pick up all the balls with one gripper, move to the next room, then drop them all. The length of this plan, the heuristic value of the initial state, is $n + 1 + n$, that is $2n + 1$ (where n is the number of balls). If, in the initial state, a ball is picked up using one of the grippers, the relaxed plan for the resulting state will be to pick up the remaining balls in the other gripper, move to the second room and then drop them all; this has a length of $(n - 1) + 1 + n$, that is $2n$, which is less than the heuristic value of the initial state so this action will be chosen as the one to apply.

The next state, however, is at the start of a plateau. The actions applicable (those for which all the preconditions are satisfied) are either to drop the ball that has been picked up, pick up another ball or move to the next room. The 'correct' action would be to pick up another ball: the relaxed plan to the goal state for the resulting state would be to drop one of the balls, pick up all the remaining balls in the newly-freed gripper, move to the next room, and drop all the balls; however, the heuristic value of this state would be $1 + (n - 2) + 1 + n$, or $2n$, the same heuristic value as the state in which the action is applied. Moving to the next room would produce a state with the heuristic value of $2n$ (return to the initial room, pick up remaining $(n - 1)$ balls, drop all balls in the final room—no move action is required to move back to the final room as the delete effect, not being in the final room, of the action to move back to the initial room has been removed). Dropping one of the balls would also produce a state with a heuristic value of $2n$ (pick up all remaining $(n - 1)$ balls in newly-freed gripper, move to next room, drop all balls). As all successor states have the same RPG heuristic value as their parent state the heuristic is unable to provide useful guidance as

to which action to apply: thus, a plateau has been encountered.

With some exhaustive search forward from this point, an improvement in heuristic value can be made in two ways: either move to the next room then drop a ball, or pickup a ball then move to the next room—both of these lead to heuristic values of $(2n - 1)$. The same plateau will, however, be encountered each time the robot is in the first room, holding one ball, and the action choices are either to pick up another ball or move to the next room (or drop a ball). Each time the plateau is encountered, the action sequence to escape the plateau is identical—move-drop or pickup-move (in EHC the actual sequence chosen will depend on the order in which the actions are considered by the planner). Having to discover one of these action sequences by exhaustive search each time the plateau is encountered is a considerable bottleneck in the search process: this is true in general for many domains.

In order to address the overheads caused by recurrent plateaux in the search space, Marvin memoises the action sequences used to escape the previously encountered plateaux; these action sequences are used to form Plateau-Escaping Macro-Actions. A macro-action is generated from the action sequence by replacing the entities in the parameters of the actions with placeholder identifiers, which then form the parameter list of the macro-action. Returning to the pickup-move action sequence, the action sequence:

0: pickup robot1 ball2 room1
1: move robot1 room1 room2

would form a macro-action:

pickup-move (?a - robot) (?b - ball) (?c - room) (?d - room)
0: pickup ?a ?b ?c
1: move ?a ?c ?d

When the search is later attempting to escape a plateau, i.e. the planner cannot find an immediate successor state with a strictly better heuristic, these macro-actions are available for application. If the plateau is sufficiently similar to one previously encountered, then the macro-actions will be able to lead the search to a heuristically strictly-better state by skipping over the intermediate heuristically-worse states. The plateau-escaping macro-actions are only used when the search is attempting to escape a plateaux—this avoids introducing overheads, slowing down search, where the RPG heuristic is able to provide effective guidance using only single-step actions.

3.4.2 Introducing Concurrency into Macro-Actions

During the process of creating macro-actions from a plan a partial-order is lifted in order to allow the independent threads to be identified. Using the partial order it is possible to introduce concurrency into the macro-actions. As long as the ordering constraints are respected, action steps the same distance from the start of a given sub-plan can occur concurrently. Reasoning about concurrency within the macro-action is only necessary when the macro-action is created: if the macro-action is used again it will already have the concurrency embedded into it. Introducing concurrency allows for reduction of the *makespan* of the plans produced, that is the time taken to complete the execution of the plan. If actions that do not interact can be placed in parallel, rather than being sequenced unnecessarily, then the makespan of the plan will be shorter despite the fact that it contains the same number of actions.

As Marvin's concurrency support is disabled when performing exhaustive search to escape a plateau, the macro-actions provide a means of reducing makespan by allowing concurrent plan segments to be inserted into the plans where, previously, a sequential plan segment would be devised. Upon escaping each plateau the new parallel plans are added to the end of the original plan rather than the original, sequential, plateau-escaping sequence.

3.5 Planning with Macro-Actions

Explicitly reasoning about macro-actions offers the advantage that they can be treated differently to other actions. More stringent pruning can be applied to macro-actions than to other actions. Pruning macro-actions will not affect the completeness of the search: the original domain description does not include these actions; they can therefore be pruned more aggressively. Macro-actions do have the potential to greatly increase the branching factor, therefore the more strict pruning is necessary.

3.5.1 Macro-Action Pruning

Many past learning approaches using macro-actions have treated these actions as a black-box. The actions are added to the domain, as given to the planner, with no indication that the actions are macro-actions; therefore all actions are treated the same by the planner. As discussed in section 2.2.1 macro-actions increase the branching factor of the search and can therefore cause search to be

more expensive if not carefully managed. Whilst many of the plateau-escaping sequences are helpful in planning, some are specific to the situation in which they were derived; a situation which might not occur again in the plan. For these reasons, macro-actions are reasoned about differently to the other actions in the domain in Marvin; this allows reasoning to be done to conclude when it is appropriate to use macro-actions. This reasoning is especially important in Marvin as macro-actions are learnt during the planning process and there is no human intuition, or large test suite, to allow reusable macro-actions to be identified.

Treating macro-actions differently also allows Marvin to exclude them from use in the relaxed planning graph during the heuristic computation. Macro-actions could potentially allow the planner to make a more accurate heuristic estimate; however, because many groundings of all the macro-actions are possible building the relaxed planning graph becomes too expensive. The relaxed plan extraction phase is also affected adversely. Macro-actions are considered in search to generate a new state, that state is then evaluated using the standard heuristic computation[1], ignoring macro-actions.

Applying Macro-Actions Only on Plateaux

During the search to solve a given problem instance all macro-actions generated earlier in the process of solving that problem instance are available for use. This means that potentially many macro-actions may have to be considered at every choice point. In order to decrease the cost of using macro-actions during search Marvin uses a strategy to prune macro-actions so that they are only used at the times when they are most likely to be useful.

Plateau-escaping macro-actions are generated from situations in which the heuristic has broken down; therefore, the heuristic can be used as an indicator of when they are likely to be useful again during planning. As areas of symmetry within the solution plan involve the application of similar (or identical) sequences of actions, they are likely to have similar heuristic profiles. In the case of plateau-escaping action sequences, the heuristic profile of the search landscape at their application is an initial increase (or no-change) of heuristic value, eventually followed by a fall to below the initial level: the profile occurring at a local minimum. If the plateau-escaping macro-actions are to be reusable it is likely that the re-use will occur when the planning process is in a similar situation. As such, they are

[1]The computation is actually slightly more sophisticated as it allows ADL actions to be used in the RPG; however the resulting heuristic value is the same and no macro-actions are included in the computation.

only considered for application when searching to escape a plateau, as this is the situation in which they are most likely to be useful. This is also the situation in which the planner most needs alternative search guidance as the heuristic is being uninformative: we avoid cluttering the search space with many macro-actions when the heuristic is able to guide the search quickly in an appropriate direction.

Using macro-actions only when necessary also helps to reduce the number of sub-optimal action sequences caused by the inclusion of macro-actions in the solution plan. Situations can arise where the use of macro-actions increases the makespan of the resulting plan due to redundant action sequences. For example, if in a simple game domain—with actions to move up, down, left or right—if a macro-action is formed for 'left, left, left, left' and the optimal action sequence to escape a given plateau is 'left, left, left' then '{left, left, left, left}, right' may be chosen if the state reached by moving left four times is heuristically better than the one reached by applying a single-step 'left' action. Such action sequences are not necessarily problematic: they can often be reduced in a post processing step by removing segments of the plan that occur between identical states. In directed search spaces, however, the redundant segments may be difficult to identify, and the remedial actions necessary to correct the plan may increase the plan length. This highlights the important balance between solution quality and techniques to increase the speed with which the problem can be solved.

Using Helpful-Action Pruning to Prune Macro-Actions

In Marvin, as in FF [32] (see section 2.3.5), only helpful actions are considered for application at each stage to reduce the branching factor (the helpful actions being those at the first time step in the relaxed plan from the current state to the goal state, plus other actions that achieve the same effects). This strategy greatly improves the performance of the enforced hill-climbing strategy. It does not affect the ability of EHC to find a solution as much as it would first appear: although the relaxed plan is not always a good estimate of the path to take to the goal, the quality of the estimate it makes deteriorates the further away from the state in which it was built one becomes. This is because in the start state of the relaxed plan the actions are not relying on any preconditions that would have been deleted by previous actions since these are the predicates achieved in a real state in the search space; the further into the relaxed plan reached, the higher the probability that the actions are relying on preconditions that would have been deleted had the delete effects of the actions not been ignored. Therefore the first steps of the relaxed plan are a more reliable prediction of the next steps in the

solution plan than the whole relaxed plan is of the remainder of the solution plan.

As helpful-action pruning greatly increase the performance of the planner it would not make sense to make many macro-actions available to the planner to negate this benefit. To minimise the effect of adding many macro-actions to the domain only macro-actions whose first step is a helpful action that is in the first step of the relaxed plan[2] are considered for application: this allows non-helpful macro-actions to be pruned, reducing the branching factor. Only those macro-actions most likely to be useful in finding a solution plan are considered. Considering only helpful actions allows the significant benefit that many macro-actions can be added to the domain and only the ones that appear to be useful can be considered at that point in the search space; rather than having to limit the planner to a small number of macro-actions.

3.5.2 Symmetric Action Pruning

Adding macro-actions to the list of actions to be considered increases the branching factor. Additional search-time pruning is therefore more valuable than ever. Marvin exploits symmetric structure in the world states arising during planning to prune the action choices at the expansion of each state during both EHC and best-first search. This pruning enables Marvin to discard groundings of a given action when another, symmetric, grounding has already been selected for evaluation. This symmetric pruning is applied to all actions, not just macro-actions, and does not affect the completeness of the search [3].

Marvin identifies symmetric objects and symmetric actions in the same manner as STAN [24]. Object symmetry groups are calculated at each state encountered during search; they are then used to prune the list of actions considered to generate the successor nodes by only keeping one exemplar for each identified group of symmetric actions. When performing EHC, the applicable action lists are first pruned to keep only the helpful actions, and the duplicate symmetric actions are then removed. When performing best-first search, the helpful actions are not used to prune the lists of actions to apply, but duplicate symmetric actions are removed from the list of all applicable actions in each state.

[2] In general helpful action pruning allows actions that achieve the same effects as these but to ensure stricter pruning these are discounted here

[3] The completeness of best-first search is unaffected; however, EHC is not complete in general but symmetric action pruning does not make EHC any less likely to find a solution

Identifying Static Asymmetry

Symmetric action pruning provides a good mechanism for pruning successors whilst maintaining completeness; however, the cost of calculating the symmetry groups at each stage becomes more significant on large, asymmetric problems. To reduce this overhead, the problem is preprocessed to establish static asymmetries. Two entities are said to be statically asymmetric if the static predicates (those that are not added or deleted by any action in the domain) pertaining to them in the initial state differ: as the static predicates are persistent, this difference will prevent the two entities from ever being functionally symmetrical.

In domains with a large amount of asymmetry, identifying static asymmetries can rule out the possibility of some entities being symmetric with any others. In such cases, the overheads incurred by calculating symmetry groups are greatly reduced as the entities that will never be symmetric with other entities are each given their own group (of which only they are a member), leaving the symmetry group calculation to establish the remaining symmetry groups considering only the potentially-symmetric entities.

3.5.3 An Example: Heuristic Landscape in the Philosophers Domain

Planning with useful macro-actions has clear advantages, particularly if the macro-actions generated are appropriate to the domain and can be used in situations where the heuristic is unable to offer appropriate guidance. Figure 3.2 shows the value of the heuristic with and without macro-actions across the solution plan generated by Marvin for a problem taken from the Philosophers domain (part of the Promela domain from IPC4) involving 14 philosophers. Initially, no macro-actions have been learnt so the search done by both approaches is identical. For the first 14 action choices the value of the heuristic, shown by the line in the graph, moves monotonically downwards as the planner is able to find actions to apply leading to strictly better states.

After time step 14, the heuristic value begins to oscillate, at this point the planner has reached a plateau: there is no state with a strictly better heuristic value that can be reached by the application of a single action. As this is the first plateau reached, no macro-actions have been generated so the heuristic profiles are identical for both configurations. At time step 25 a state is reached that has a better heuristic value than that at time step 14; it is at this time that the plateau-escaping macro-action will be generated, memoising a lifted version of

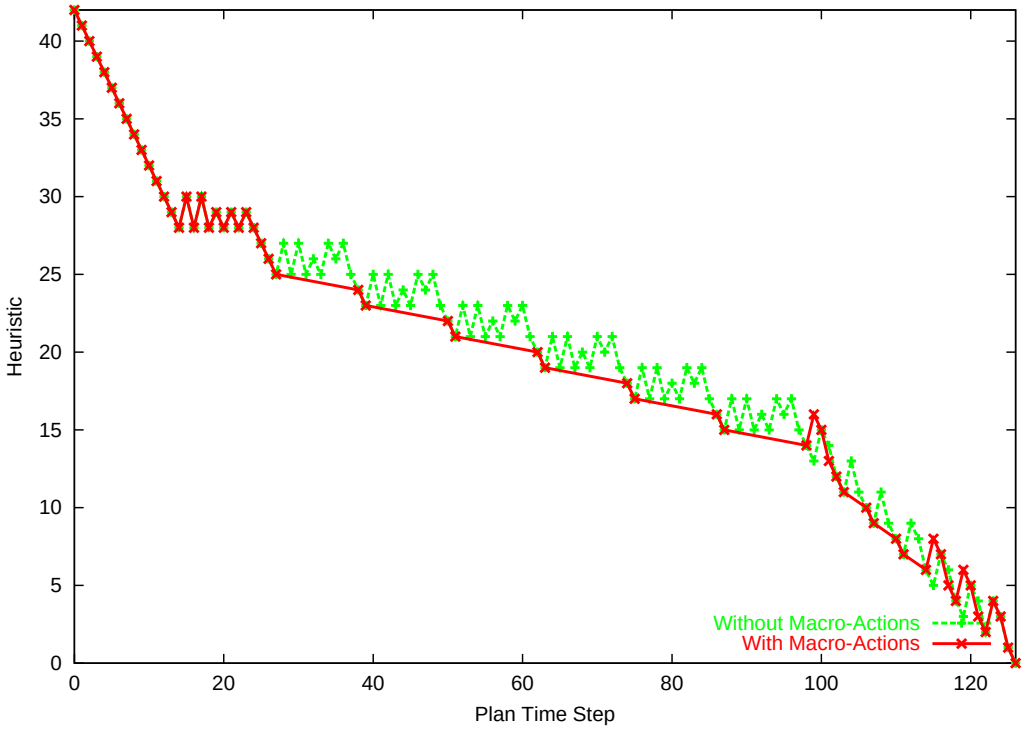


Figure 3.2: Heuristic Landscape During Search to Solve a Problem in the Philosophers Domain

the sequence of actions that was used to escape the plateau. A brief period of search in which a strictly better state can be found at each choice point follows before the planner again hits a plateau.

The subsequent six plateaux consist of applying the same sequence of actions to each pair of philosophers: it can be seen that the heuristic fingerprints of the plateaux are identical. The version of Marvin in which macro-actions have been disabled repeats the expensive exhaustive search at each plateau: the heuristic value again goes through the process of increasing and then decreasing before reaching a strictly better state as FF would. The version using the plateau-escaping macro-actions, however, now has a single action to apply that achieves a strictly better state and search continues, stepping over the subsequent plateaux through the selection of macro-actions that yield strictly better states.

When all of the larger plateaux have been overcome a series of smaller plateaux are encountered. Again, it can be seen that, for the first of these, both versions must complete a stage of exhaustive search; however, after the first of the smaller plateaux has been completed, the macro-action formed allows the subsequent plateaux to be bypassed. Finally, the plan finishes with a previously unseen sequence of actions, which both versions must do exhaustive search to compute.

During the search to solve instances of the philosophers problem using macro-actions a large proportion of the time is spent escaping the first plateau and the remainder of the plan is computed much more quickly. As the instances become larger the time taken to complete each stage of exhaustive search grows dramatically and the time saved through using macro-actions becomes greater.

3.6 Chapter Summary

This chapter has introduced the features of the planner Marvin, highlighting the differences between it and the well-known planner FF upon which it is built. An understanding of the workings of Marvin is important, both in understanding the framework in which the new macro-action generation techniques are implemented and in interpreting the results presented in chapter 5.

An important new technique has been introduced: the generation and use of plateau-escaping macro-actions. These macro-actions are built from sequences of actions used to escape plateaux, areas of the search landscape where the heuristic is uninformative. The parameters of the macro-action are lifted to generalise it; that is, instead of being bound to specific entities they become variables that can be bound to any entity of the appropriate type. In memorising such action sequences later episodes of exhaustive search can potentially be avoided by applying the single macro-action if the same sequence of actions is required later during search. This technique is a novel approach to macro-action generationit differs from other approaches in two important ways. First, it takes inspiration from heuristic weakness; and second it generates macro-actions online, during the planning process, without the need to solve additional training problems.

Chapter 4

Management and Use of Macro-Actions in Planning

This chapter details the techniques implemented in Marvin to deal with the management of a library of plateau-escaping macro-actions. This includes techniques for the extraction and use of plateau-escaping macro-actions using different heuristics and strategies for simulating the use of macro-actions through action reordering.

4.1 Caching Macro-Actions

Having solved one problem in a given domain it would be wasteful to disregard everything that has been learnt in doing that when going on to solve another problem in the same domain. Many of the plateau-escaping macro-actions provide a useful insight into the structure of the search landscape for a given domain, under the RPG heuristic, not just for a single problem in that domain. It would therefore be useful to store these macro-actions for use in solving future problems. In doing this, however, the issue of increasing the branching factor in the search space becomes even more critical. The branching factor will not only be increased by macro-actions discovered during the search to solve each problem; but by those generated at any plateau found during search to solve any previously seen problem in the domain.

Upon solving each problem all plateau-escaping macro-actions are stored in a library. Macro-actions are represented as sequences of unit actions with corresponding parameters labelled; rather than compiling out preconditions and effects to treat them as one action. The information that is collated about each macro-action during search is stored in the library with the macro-actions. The

information that can be found is the number of times a macro-action has been used in the solution plan; the number of times the macro-action has been instantiated; information about the heuristic profile at the point the macro-action was generated; and finally the number of problems that have been solved since the macro-action was last used. This is the only additional information available about each macro-action that can be generated using this online learning strategy. Macro-actions in the library are ordered according to their use count.

The use count of macro-actions is not simply incremented only if the macro-action is used in EHC. Doing this would disadvantage macro-actions that are used to solve a problem by best-first search and could potentially have been used in EHC to avoid resorting to best-first search. Instead, macro-action use counts are updated upon solving problems; this ensures that only macro-actions that appear in successful solution plans have their use count updated. The plan is post processed, analysed to identify which action sequences represent plateau-escaping sequences and independent action sequence threads are extracted from the identified sequence. When the action sequence threads have been identified the use count of the corresponding macro-action in the library is updated, if one exists; or a new macro-action, corresponding to this sequence, is generated if no such macro-action exists. This strategy also ensures that the same macro-action cannot be entered into the library twice. When macro-actions are pruned from the library they are completely removed and do not remain in the library; thus, if they are regenerated and entered into the library again their previous use count will not be maintained, instead counting will resume again from zero.

In order to successfully generate plateau-escaping macro-actions the planner must be able to escape the first plateau that it encounters. If the planner has to perform exhaustive search on a small problem the absolute increase in time to solve that problem will be smaller than if it had to solve a larger problem through exhaustive search. It can therefore be seen that the approach of caching macro-actions is sensitive to the ordering in which the planner is presented with the problems. The total time taken to solve a whole collection of problems will be greater if the macro-actions must be discovered on the harder problems (i.e. the harder problems are presented first) and the number of problems solved may be decreased if a time limit is imposed for solving each problem.

As a large library of macro-actions is generated a good pruning strategy is essential. If the planner were to store all macro-actions generated during the process of solving all problems, and then consider their application at every point during the search the performance of the planner would almost certainly degrade

significantly due to the increased branching factor.

4.2 Managing a Library of Cached Macro-Actions

Many machine learning techniques work by solving the problems with and without using a given feature, in this case a macro-action, and compare the results obtained from both tests to decide whether or not that feature is beneficial. This is offline learning requiring extra time and search effort in order to generate macro-actions before problem solving can begin. In the approach presented here, however, the goal is to do the learning online, without the need to solve additional problems, a different strategy is therefore required. Online learning has two major benefits in this approach, the first being the removal of the requirement to solve additional problems; the second is that learning can be done not only on small instances (which may not accurately represent the larger, more interesting, problems) but also on the larger instances giving information that is potentially more useful.

4.2.1 Search-Time Pruning

The most obvious approach to library management is simply to keep all of the macro-actions ever generated. The advantage of this is that no useful macro-actions will ever be pruned as a result of an overly aggressive pruning strategy. The disadvantage is the library of stored macro-actions will grow indefinitely and macro-actions which are of no use will be kept and may have to be considered during search. This has the potential to greatly increase the branching factor and make search to find a solution to the problem considerably more difficult.

The only reason that it is possible to even consider this approach to macro-action pruning is that the search behaviour of Marvin ensures that some macro-action pruning is done *during* search in addition to that done in the library (see section 3.5.1). By considering macro-actions after all other actions in the domain, and by using only helpful macro-actions it is possible to reason with a large collection of macro-actions without the same increase in cost that would otherwise greatly inhibit the progress made by search.

Keeping all the generated macro-actions and relying on Marvin's search behaviour is one possible strategy; it may, however, be possible to achieve better performance by pruning the library of macro-actions itself based on the experience

of solving the previous problems.

4.2.2 Survival of the Fittest

The survival of the fittest approach is similar to that taken by many offline learning techniques, keeping the best performing macro-actions, but in this approach the results of running the planner with and without the macro-actions are unknown so this comparison cannot be used to generate a fitness function. The fitness function is instead the number of times that a given macro-action has been used in the past.

The macro-actions generated are actions that allow the planner to 'patch' the heuristic; rather than those simply to speed up search in arbitrary places. It is, therefore, more likely that the use of a macro-action implies better performance than would occur in not using it; in using the macro-action some exhaustive search has been avoided. This is not an assumption that can be made in general when using arbitrary macro-actions: the potential gains from using these macros are often not as great. The use of other macro-actions may simply indicate that they are widely applicable and not that any improved search performance resulted: the heuristic may well have quickly found the correct path anyway. To evaluate the fitness of plateau-escaping macro-actions it is therefore more reliable to use the metric of number of times used; for other macro-actions their use is not as likely to imply any great time improvement so they must be evaluated by comparing the time taken to solve problems with, and without, macro-actions. This assumption will, of course, be tested when the survival of the fittest strategy is evaluated.

The pruning strategy employed is to keep only the n most used macro-actions. The larger the value of n is, the more the branching factor is potentially increased; the smaller the value of n the greater risk of discarding useful macro-actions. Macro-actions that are useful in many problems will be allowed to persist in the library, brief periods of a macro-action not being used will not necessarily cause widely useful macro-actions to be discarded. This approach also imposes an upper limit on the size of the library; this means that the increase in branching factor is more tightly controlled but the best macro-actions can still be kept.

4.2.3 Time-Out Pruning

The motivation for time-out pruning is to reduce the size of the library of macro-actions by removing those macro-actions which have not been useful in solving recent problems, on the premise that they must therefore no longer be useful.

The basis of this strategy is to delete macro-actions that have not been used in solving the last n problems from the library. Some macro-actions are never re-used after being discovered once: such actions will be removed from the library reasonably quickly, thus only being present increasing the branching factor in a few problems. Pruning these macro-actions allowing a reasonably short life (i.e. using a small value of n) will help to prevent the situation where large numbers of useless macro-actions generated across many problems will be present.

The success of this strategy will clearly depend on the decided value of n, the *caching interval*. If the caching interval is very short, i.e. n is small, then potentially useful macro-actions may be discarded too hastily; if the caching interval is too large then it is likely that the planner will have to deal with large numbers of non-reusable macro-actions. It is worth noting at this point that if any useful macro-actions are discarded they may well be rediscovered: if the planner encounters an identical plateau in a later problem the macro-action will be re-entered into the library. Search effort will be required to rediscover the action, but problems following the re-discovering problem *may* this time allow the use of this macro-action. This would allow the macro-action to survive longer, if appropriate, when regenerated. There is, of course, a trade off between keeping macro-actions for a large number of problems in case they may be useful thus slowing down search in those problems; and solving a later problem faster because the macro-action is still present. The pathological case for this would if n took the value 2 and the problems had a repeating structure, in the order presented, such that a macro-action was only required in every third problem: it would, of course, then have to be regenerated every time it was needed.

The advantage of using the time-out approach to pruning is that, if the problems are presented to the planner in order of difficulty, any phase transitions, causing different macro-actions to be useful, can be accounted for. It may be the case that a macro-action is useful in solving many easier problems; but in harder problems the same macro-action may not be useful due to some fundamental difference in the structure of the problem. Given this pruning strategy a macro-action will be removed from the library, regardless of the number of times it has been used, when it has not been useful in solving recent problems. Thus, any macro-action that was very useful in easier problems, but is not so useful in harder problems, will be deleted from the library and will not needlessly increase the branching factor when the planner is solving harder problems.

4.2.4 Roulette Selection from the Macro-Action Library

Another approach is to keep all of the macro-actions in the library and then select just a few from that library at each choice point. It is desirable to bias the choice of selected macro-actions towards those macro-actions that are the most likely to be useful. In order to allow newly generated macro-actions the opportunity for selection, however, it is desirable to allow the possibility of selection for each macro-action. Ensuring that newly generated macro-actions have the opportunity for selection will allow the discovery of other potentially useful macro-actions. The basis of this strategy is to keep all the macro-actions ever generated and then roulette select between macro-actions, the bias being given by the number of times the macro-action has been used plus one[1]. The selection process occurs at the expansion of each state in the search space. If a macro-action is selected and used its use count is incremented accordingly.

4.2.5 Pruning Based on Instantiation Versus Usefulness

Macro-FF [7] uses what is referred to as a dynamic pruning strategy. The strategy is not dynamic in the same way as the pruning strategies discussed here: pruning is simply done once after the solution of n real problems. Further the pruning takes place on a library of already statically pruned macro-actions. The basis it uses for its dynamic pruning strategy is the ratio between the number of times a macro-action has been used and the number of times it has been instantiated. The reasoning behind this is linked to the trade-off when using macro-actions between the potential cost they incur in increasing the branching factor and the potential for speeding up search by making several decisions at once.

Macro-actions only add additional search costs to the planner when they are instantiated, thus a macro-action that is not instantiated many times is not going to adversely affect the performance as much as one that is instantiated many times. The number of times a macro-action is instantiated is dependent on how frequently its preconditions are satisfied in the states reached during the plan and how often it is pruned by other search-time pruning strategies, such as helpful macro-action pruning. The purpose of using macro-actions is to improve search performance, it is therefore sensible to consider the potential use of a macro-action as well as limiting the potential for negative impact on performance. It may be that a certain macro-action, although instantiated many times, is selected as the correct action to use many of the times it is instantiated; whereas an action that is

[1]The reason for adding one is to ensure that the macro-actions that have never been used have some chance of selection.

not instantiated a large number of times may not be used at all, thus adding some additional search cost without giving any benefits. The major topic of interest here is the net benefit gained from using that macro-action. Again, since the planner is only being ran once on each problem instance there is no indication of how the performance would vary with or without using the macro-actions. In Macro-FF, since the macro-actions had been statically pruned, the assumption was made that the use of a macro-action had a positive effect; as before, for the purposes of the pruning strategy, it is necessary to assume that the use of a plateau-escaping macro-action implies a performance improvement. The success of the caching strategy will clearly depend on this assumption.

This strategy is a development of that used in Macro-FF to make the pruning truly dynamic and to use plateau-escaping macro-actions. The macro-actions used here are not actions that have been statically pre-filtered, and are therefore not a small collection of macro-actions already tested for potential usefulness. The dynamic pruning strategy is used after the solution of every problem in search, with new macro-actions being added to the library after solving every problem. Macro-FF could not do this as it only generated macro-actions in its training step, so pruning after every problem would simply result in the library size decreasing with no potential for macro-actions to be replaced.

4.2.6 Heuristic Profile Based Selection

The statement that a state is on a plateau is, in fact, a generalisation of a number of possible heuristic landscape properties that could be encountered during search. In his paper discussing search landscape topology Hoffmann [34] describes several types of plateaux[2]. His analysis was, however, one of the complete search space, not all types of plateaux can be characterised without doing this (clearly analysing the whole search space is not a practical technique to use during search).

Two types of plateau can be identified by Marvin at search time:

- **Local Minima:** when all successor states (generated by the application of unit actions) have been evaluated and all have worse (higher) heuristic values than the current state.

- **Saddle Points:** when all successor states have been evaluated and have

[2]Hoffmann's definition of plateau is specifically a region where all states have the same heuristic value; here, a more general definition has been used in which plateaux are regions with heuristic value the same or no worse than a given starting state (the state reached by applying the actions in the plan so far).

a heuristic value worse than or equal to the heuristic value to the current state.

Plateau-escaping macro-actions will be generated from states where the following plateau corresponds to each of these categories. When deciding which plateau-escaping macro-action to apply the planner is attempting to characterise the state as similar to the state in which the macro-action was generated. Selecting only the macro-actions generated from states on the same type of plateau on which the planner is currently situated may allow the planner to identify the correct macro-action to use (if one exists). The library is sorted by macro-action use count, as in the other strategies, but only macro-actions that correspond to the same plateau type the planner is currently on will be considered during search. The same macro-action may, of course, be generated at both a saddle point and at a local minimum, in which case the macro-action will be allowed to be used in both situations.

4.3 Generation and Use of Plateau-Escaping Macro-Actions Under Other Heuristics

Although this technique has been developed and discussed primarily using enforced hill-climbing under the RPG heuristic the idea can extend to other types of planning technology using different heuristics. All heuristics have a weakness [3] and the weaknesses occur at different points. The technique is generic and can be applied to different heuristics by taking the sequences of actions that correspond to plateaux under each heuristic and applying the same macro-action generation and management techniques used under the RPG heuristic.

4.3.1 Generating Plateau-Escaping Macro-Actions for the Causal Graph Heuristic

The planner diagonally downward [29] demonstrated excellent performance in many domains in IPC 4 [31]. It used a novel heuristic, the causal graph heuristic, in combination with the RPG heuristic, to perform best-first search (see section 2.3.7). It is possible to use the causal graph heuristic in enforced hill-climbing, the results of doing so have been published [28]. When performing EHC search

[3] unless solving the exact problem is used, which is clearly nonsensical since there is no need for a heuristic once the problem has been solved.

with the causal graph heuristic plateaux are encountered, as they are when using the RPG heuristic. There is, however, a difference in the heuristic landscape of the search space: the two heuristics are different and will lead search in different directions, as well as evaluating states differently. Plateau-escaping macro-actions can be generated under the causal graph heuristic in the same way that they can be generated using the RPG heuristic. The macro-actions generated are likely to be different to those generated under the RPG heuristic, due to the differing heuristic profile of the search landscapes. The macro-actions generated under the causal graph heuristic can be used in further search to solve the problem under that heuristic, and cached for use in future problems, in the same way those generated under the RPG heuristic. Depending on the number of plateaux encountered and their similarity the technique may be more successful under one heuristic than the other.

4.4 Simulating the use of Macro-Actions Through Action Reordering

In using macro-actions a trade-off is introduced between the increased branching factor due to the extra actions added to the domain and the potential decrease in solution length. This trade off determines number of states that must be considered during search and therefore whether any benefits are gained by using macro-actions. Macro-actions encapsulate a useful sequence of actions taken from the domain: they represent the observation that it is often, but not necessarily always, a good idea to select a certain action following another action. Another way to model this approach, rather than adding a macro-action to the domain, is to reorder actions so that the first action considered at the next choice point after applying a given action will be intelligently selected based on an idea of which action is likely to be successful following that action.

If the next action predicted by the ordering strategy is indeed added to the plan the search effort saved is almost equivalent to having applied a macro-action in the previous state. Given a correct action reordering for a given action pair only one action will be considered at the next choice point, in EHC, to select the second action. This requires only one state to be generated and one heuristic computation. Only one more state will be evaluated, if this is the correct action sequence, than if the equivalent macro-action had been applied at the earlier point in search.

This approach does not have the disadvantage of increasing the branching

factor, since no actions are added to the domain. Of course, reordering actions can potentially result in more search effort being required to solve the problem if the action ordering strategy considers the correct actions after than the original action ordering strategy would have done. Since many planners, including Marvin, use an arbitrary ordering, usually that in which the actions are presented in the domain, it is unlikely that any additional insights are being lost by altering this action ordering. The only additional cost incurred is that of sorting the list and maintaining which actions should follow other actions.

4.4.1 Probabilistic-Observation-Based Action Reordering

Macro-actions that are used frequently represent sequences of actions which follow each other often. An action ordering strategy based on the number of past occurrences of a given action following another action could be used to implicitly suggest the macro-action to the planner without actually adding the macro-action to the domain. By noting the number of times a given action follows another in the plan and ordering actions based on this it is possible to simulate the addition of macro-actions to the domain without actually having to add the additional actions. For example, the macro-action pickup-move-drop in the gripper domain may be extracted by a macro-action-generating planner due to frequent occurrence, being a plateau in the search space, or through improving search performance. It is also possible to simply observe that so far during search to solve a problem, and indeed in search to solve previous problems, every time a move action has been applied that a drop action was applied afterwards, thus it would be sensible to consider the drop action after the move action has been applied in future search. Similarly the move action may have followed the pickup action 50% of the time; the other 50% of the time the pickup action is likely to have been followed by another pickup action. It is therefore sensible to consider applying the move action and the pickup action following the application of the pickup action before considering all other actions.

The basis of this strategy is, therefore, to order the list of potential action choices by an estimate of the probability that they will follow the last chosen action in the plan. The probabilities are not used in the traditional sense of being normalised to have a sum of 1 since there are a different number of actions at any given choice point. All the probabilities for every action following another action are initialised to zero representing the frequency with which an action has been applied following another. As planning progresses each time a given action follows another one its following frequency count is increased. When a node is

to be expanded in search the possible (helpful) actions to generate successors are ordered according to this frequency metric and their corresponding successor states are expanded in the given order. Since EHC will chose the first possible action that leads to a state with a strictly better heuristic value if the first action chosen leads to such a state a great deal of search effort will be saved.

4.4.2 Identifying the Most-Likely Preceding Action

Plans generated by an FF-style planner often have independent threads of execution interleaved. For example, in a logistics-style domain it is possible that a plan might have interleavings of the actions to perform tasks in several distinct locations. The planner may have the plan (load package1 truck1 location1) (drive truck1 location1 location2) (unload package1 truck1 location2) and a similar sequence of actions for another package in another location: (load package2 truck2 location3) (drive truck2 location3 location4) (unload package2 truck2 location4); interleaved with the end of another earlier sequence concluding with (unload package3 truck3 location5). These plans could be interleaved arbitrarily to create the overall plan, an example is given in figure 4.1.

Updating the probabilistic following data to reflect the interleaved plan could be misleading: for example, it would state that load followed by unload was a promising action sequence for the planner to consider in the general case. Although the interleaving could occur in this way again, it is largely arbitrary and it is not the case in general that it is good guidance to follow a load action with an unload action: indeed if the two actions share a common package parameter it is actually an unwise action choice, simply undoing the previous action and introducing redundancy into the plan. The strategy of updating the previous action in the plan regardless of threads of execution that may be present is referred to as U1 in the evaluation. Further reordering strategies have been developed to take into consideration only those actions that appear in sequence and are part of the same thread of execution. This is related to the work by Botea [6] in the version of Macro-FF using component abstraction: related objects are detected (by identifying static predicates pertaining to both objects) and macro-actions are made that represent activity restricted to operations that involve parts of this collection of objects.

0:(load package1 truck1 location1)
1:(unload package3 truck3 location5)
2:(drive truck1 location1 location2)
3:(load package2 truck2 location3)
4:(drive truck2 location3 location4)
5:(unload package1 truck1 location2)
6:(unload package2 truck2 location4)

Figure 4.1: Example Plan Segment for the Driverlog Domain

Update if the Previous Action Shares A Parameter

In the first of these more sophisticated strategies, U2, probabilistic data is only updated if the last action added to the plan shares a parameter with the action that is to be applied immediately before it. Actions sharing a parameter are reliant on the some common object in the world and so can be considered linked as part of a sequence of execution. Updating the probabilities for only those actions that share parameters allows the cached data to better represent real interdependencies between actions.

A problem with this strategy is that in only updating the action following data if the action added is related to the last action in the plan, information is potentially lost about the action that a given action actually followed from. Suppose, during the planning process to find a solution to a driverlog problem actions are added to the plan in the sequence shown in figure 4.1. Using this update strategy the data that unload follows load (steps 0-1), drive follows unload (steps 1-2) and load follows drive will, correctly, not be added since the parameters are pairwise unrelated. Between steps 3 and 4 the fact that drive followed load will be noted as these actions share a parameter. Between steps 4 and 5 no information will be added since the parameter sets do not overlap; Similarly, no information is recorded between steps 5 and 6. The fact that unload follows drive would not be recorded anywhere in that sequence due to the interleaving of the plan. Looking at the independent threads; however, it can be seen that unload has, in fact, followed drive twice in the individual sequences of execution in the plan.

Update Based on Last Non-Mutex Actions In the Plan

To address this issue the final strategy, U3, has a slightly more complex update rule. Instead of simply looking at single preceding action at the end of the plan, all actions at the end of the plan that are not mutex with each other are considered.

The last non-mutex actions in the plan represent all actions which could equally have been placed at the end of the plan: there is no ordering constraint between them. An action added to the plan could be following on from any one of these actions at the end of the plan; not just from the action that happens to be ordered last. To identify these potential predecessors the algorithm reverses through the plan until an action is found that is mutex with any of the actions that follow it within the plan. When any pair of mutex actions is found in the plan the search for a predecessor is stopped. The first of the pair of mutex actions is not considered as a predecessor since it precedes another action.

Consider the addition of the final two unload actions to the plan in the example in figure 4.1. Upon adding the action (unload package1 truck1 location2) at step 5 to the plan the algorithm regresses through the plan to find possible predecessors. The previous action, plan step 4, is added as a possible predecessor; the action at time step 3 is mutex with the action at time step 4 so the regression stops at this point: this action could not be placed at the end of the plan. Since none of the actions in the set of predecessors share a parameter with the newly added action no update is made to the probabilistic data. Search then continues and the action (unload package2 truck2 location4) is selected for addition at time step 6. Regressing through the plan from this state the actions at time step 5 and 4 can be added to the set of possible predecessors, the algorithm stopping at, and excluding, the action at time point 3 as it is mutex with that at time point 4. At this point the set of possible predecessors is checked for any action that could preceded the action at time step 6 in the same sequence of execution; that is, any action in that set that shares a parameter with the newly added action. The action at time step 4, (drive truck2 location3 location4), is in the set and has parameters in common with the last added action so the probability that an unload action follows a drive action is updated accordingly.

4.4.3 Cached Probabilistic-Observation-Based Action Reordering

An extension of this strategy is to cache the table of probabilistic data between runs of the planner. The data from previous runs of the planner, in the same domain, can be used to suggest action orderings for the choices when solving subsequent problems. Marvin can then build up a potentially useful table of action ordering probabilities based on the previous experience. Unlike the macro-action caching libraries, the size of these tables does not increase as more problems are solved. The information stored in the table simply becomes a more accurate

refelection of the action ordering observed planning process. There is, therefore, no need to employ any library management or pruning strategies in order to maintain the library.

4.4.4 Making Use of Action Following Data on Plateaux

The action reordering strategies discussed will, if suggesting the right action to perform, improve the efficiency of planning using EHC when search is not on a plateau. In the case where the planner simply finds a better state and then selects that without considering all other possible successors (i.e. problems with no plateaux) great improvements could be seen. Selecting the correct action to apply first would result in the planner solving the problem whilst only having to evaluate the a number of states equal to the length of the plan.

When plateaux are encountered, however, the planner resorts to exhaustive search to find an escape path: that is, that even if the planner does select the 'correct' action first it will not be obvious that this is the case as the heuristic computation cannot detect this and exhaustive search will continue until an escape route is found. Assuming a perfect action ordering strategy (i.e. one that suggests the correct action to apply at every point in the plan) the performance of breadth-first search would be very poor compared to the performance of a strategy that made use of the data. The performance of breadth-first search with the appropriate action ordering would be slightly better than the performance of breadth-first search without such an ordering but would not be improved as much as that of a search guided by this strategy.

In order to gain the maximum benefits from the action reordering data on plateaux the order in which states are visited during exhaustive search to escape plateaux is altered. The RPG heuristic is not used on plateaux, as in standard FF; but instead of performing breadth-first search, the planner performs best-first search using only the action probabilities as guidance. A best first search guided by these probabilities will allow the planner to exhibit better behaviour on plateaux if the suggested action ordering strategy is correct. The states are weighted according to their probability of being correct relative to the state at the start of the plateau. Probabilities of states that are immediate successors of starting state are equal to the probability that the action is the correct action to follow that applied to reach the plateau. Probabilities of subsequent states are the product of the probabilities of all actions applied to the state at the start of the plateau to reach the state. With equal probabilities this approach would simply be breadth-first search.

4.5 Chapter Summary

In this chapter a number of techniques for the online management of a library of cached macro-actions have been introduced. Library management is based on the features of macro-actions that can be maintained online, that is without having to solve additional training problems in order to determine the usefulness of a macro-action. The features considered are: the number of times a macro-action has been applied in solving problems; the number of problems that have been solved since a macro-action was last used; the ratio of the number of times a macro-action has been used divided by the number of times it has been instantiated; and the heuristic profile at the point of macro-action generation. Pruning techniques based on each of these features have been discussed and will be evaluated in chapter 5.

The simulation of the use of macro-actions through action reordering has also been discussed. The motivation for this is to avoid the increase in branching factor that occurs through adding extra macro-actions to the search landscape. Macro-actions can be simulated by altering the order in which actions are considered for application during search. The action ordering is determined according the occurrence frequency of an action following another action in solving planning problems. This data, like that for macro-actions, can be used throughout planning to solve a given problem as well as being stored for use on future problems. Unlike macro-action libraries, however, there is no need for pruning of this data as it is simply updating a table, the information becomes more accurate but does not consume any more space or take any longer to consider. Different strategies for identifying which action is the most appropriate to update have been considered; ranging from simply updating the preceding action in the plan, to more sophisticated strategies attempting to identify independent threads of execution using shared parameters to identify dependencies.

Chapter 5

Results

In this chapter the results of experiments designed to test the techniques discussed are presented. The main focus of the chapter is to evaluate the techniques used relative to a control version of Marvin to analyse their effects on planner performance. Extensive comparison of the planner Marvin to other state of the art planners was made in IPC 4, the results of which have been published [31].

5.1 Description of Domains Used and Heuristic Landscape Discussion

This section introduces the domains used for evaluation and discusses briefly the properties of these domains with respect to the RPG heuristic. The domains discussed include many of the domains in the two most recent competitions that were available in a purely propositional form (i.e. no metric or temporal features). Full descriptions of the domains are given in [49] (IPC 3 domains) and [31] (IPC 4 domains). The complexity of planning in each domain, taken from [30], is included in the description given here in order to select a number of domains with different properties and complexities.

If using plateau-escaping macro-actions is to be successful certain properties of the domain must hold. The required properties are that there must be plateaux in the search landscape, and that the some of the plateaux must be sufficiently similar: that is, the same (lifted) sequence of actions must useful in escaping the plateaux. Figures 5.1 and 5.2 show the heuristic profile across the solution plan for one problem taken from each of a range of domains used in IPC 3 and IPC 4 respectively. Each of the problems in a given domain has a different heuristic profile, however there is similarity between the profiles that arise in each of the

problems. It is not practical to include the heuristic profiles for all problems in the benchmark suite for every domain. In figures 5.1 and 5.2, therefore, a single representative problem has been taken from each domain for illustration purposes. The problems chosen for illustration are those with approximately 100 actions in the solution plan: this allows for a sufficiently long plan to show interesting information; while allowing the heuristic landscape to be seen clearly. Displaying plans with too many actions does not allow the finer details of the heuristic landscape to be seen in the graph. The discussion of the heuristic profiles encountered in the domain is, however, based on the profiles for all problems solved by Marvin with concurrency disabled and without using macro-actions.

The landscapes in figures 5.1 and 5.2 are generated by running Marvin without macro-actions and analysing the RPG heuristic value at each state in the solution plan. This clearly only gives the heuristic profile of the domain encountered during EHC search to solve the solution plan and is not a complete analysis of the search landscape. It is, however, an analysis of the parts of the search space that Marvin must explore, specifically the states that occur in the solution plan, in order to find the solution plan it generates; this is sufficient to aid in analysing the results. A full analysis of the plateaux in the domains would not be possible due to the size of the search space; Hoffmann [36] performed a detailed analysis based only on the small problems in a number of domains (see section 2.3.5).

5.1.1 IPC 3 Domains

The FreeCell Domain

The FreeCell domain is based on the well-known card game. The game has cards laid out in columns in an arbitrary order, the goal is to have all of the cards from each suit stacked in ascending numerical order. The task is made difficult by constraints: the only card that can be placed on top of a given destination card is one of the opposite colour with numeric value one less than the destination card. There are four free cells, from which the game takes its name, in which cards can temporarily be stored to allow easier movement of other cards. The game gives rise to an interesting feature when encoded as a planning domain; that is, it gives rise to unrecognised dead ends in the EHC search space under the RPG heuristic. Once a card has been taken off another card and put in a free cell (recall that all cards begin arbitrarily ordered in columns) it is not necessarily the case that the card can be returned back to its original position, the decision to place cards in their final position on the stack for the goal is also irreversible. The relaxed

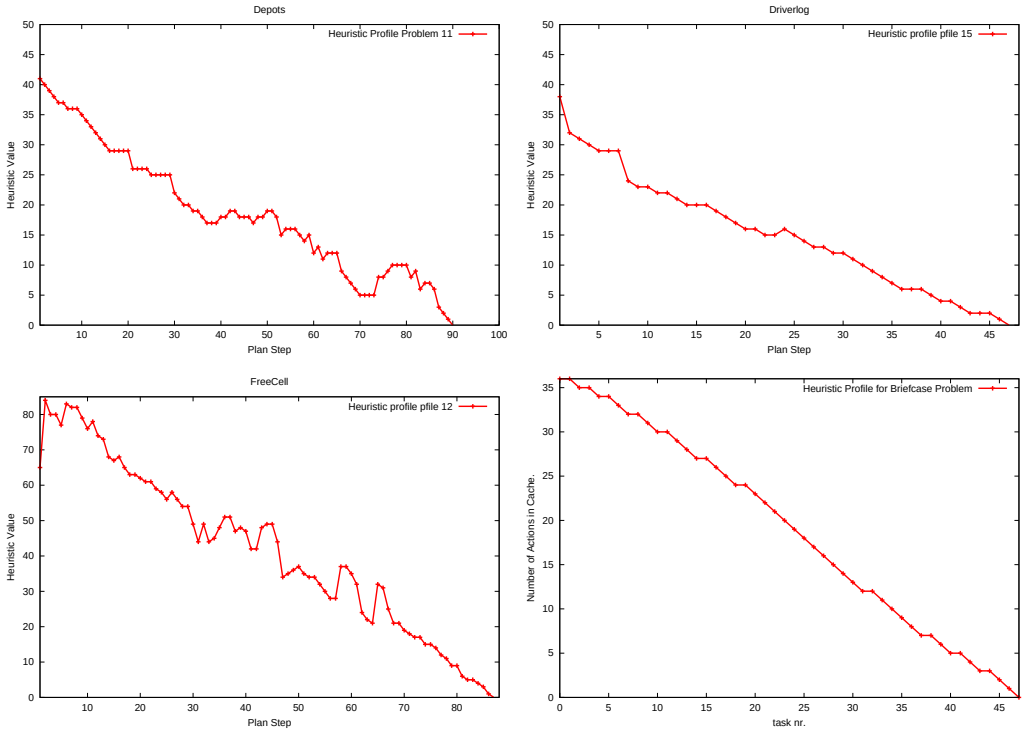


Figure 5.1: Heuristic profile across the solution plan for a problem selected from each of the IPC 3 evaluation domains (from left to right: Depots, Driverlog, FreeCell and Briefcase).

plan estimate ignores delete lists, and thus cannot see the problem that putting a card in a free cell causes the resource to no longer be available for use with other cards. In making a misguided decision the planner can reach a dead end and be unable to revert the action choices it has made.

The heuristic profile seen in this domain is irregular with many different plateau types occurring and several long plateaux. Using this domain will be an interesting test of whether the planner can select from a large number of macro-actions a small number of those that are reusable. This domain has been chosen for evaluation due to the directed nature of the search space. Theoretically macro-actions can potentially be unhelpful in directed search spaces, under EHC, by leading the planner into inescapable dead ends. This domain would therefore be expected to be one in which macro-actions may not be able to improve search performance, or even make performance worse.

The Depots Domain

The Depots domain models the movement of crates between Depots, when the crates arrive at the domains they can be stacked on top of each other, this is effectively a combined version of the logistics and blocksworld domains.

The difficulty of solving problems in the Depots domain varies dramatically

across the IPC3 evaluation suite. Some of the problems have a profile that appears similar to that shown in figure 5.2 for the Satellite domain, but with some of the step-like plateaux requiring two or three actions for escape. Other problems in the domain have a much more complex profiles, like that shown in the same figure for the Depots domain itself, with long plateaux that are difficult to escape. This domain will give a useful insight into whether the information learnt on easier problems in the domain can assist in solving the harder problems. Planning in this domain is in P, optimal planning in this domain is NP-equivalent [30].

The Driverlog Domain

Driverlog is another IPC3 domain that models a logistics problem. A location map can be specified with the additional information about which drivers, packages and trucks are present at each location. Trucks can be driven between locations and drivers can walk between locations. Typically the goal of the problem is to transport packages to a specified location. The strips version of the problem does not take into account the distances between locations; all roads between locations are considered to be equal in length.

The plateaux encountered in the Driverlog domain are varied in size. Some plateaux require a long sequence of actions for escape; whilst others require just two actions. The presence of varied plateaux will mean that many macro-actions may be generated in this domain and only some will be reusable. This will again offer a good test of whether the planner can select useful, reusable macro-actions from a collection of macro-actions containing many unuseful macro-actions. Planning in this domain is in P, optimal planning in this domain is NP-equivalent [30].

5.1.2 IPC 4 Domains

The Airport Domain

The Airport domain taken from IPC 4, models the scheduling of planes moving around Munich Airport. Planes must be manœuvred around the airport from the gates to the runway, in order to take off; and from the runway to the gates in order to reach their parking location. Constraints are imposed to ensure that no planes become too close to each other. In the heuristic profile shown for this domain, a number of plateaux can be seen; these are, however, generally reasonably short and do not constitute the main difficulty in planning in this domain using EHC. The main difficulty arises in avoiding the deadlock situation that frequently arises when two planes become too close to each other. If the planner successfully solves

the problem without having to resort to best-first search (i.e. does not reach a deadlock situation) the problem is solved quite quickly; if the planner is forced to resort to best first search the problems are generally very difficult to solve.

This is the only one of the evaluation domains in which planning is PSPACE-hard [30]. It is therefore an interesting domain to include due to being a unique exemplar of a PSPACE-hard problem amongst these domains. Further it will allow investigation of the effect of macro-actions on the interesting deadlock situation; which may often force a planner using EHC to resort to exhaustive best-first search.

The Philosophers Domain

The second domain to be considered is the Philosophers domain. This domain is the first of two IPC 4 domains based on model-checking communication problems. The Philosophers domain requires planners to perform model-checking on solutions to the well-known Dining Philosophers problem: specifically finding deadlock states that could arise.

The heuristic profile for problem 8 in this domain, shown in figure 5.2 very clearly shows plateaux which have identical heuristic profiles. Initially search begins finding a strictly better state with the application of one action. After ten actions have been applied a state is reached from which no strictly better successor state can be reached by the application of a single action. This state is the start of a plateau, it can be seen from the profile of the solution plan that there are several states with oscillating heuristic values before a better state is reached at plan step 20. One strictly better successor is found before another, different, plateau is reached. Upon analysis of the solution plan it can be seen that this second plateau, and subsequent plateaux up to and including the fourth plateau, require the same sequence of actions to be used to escape as the first plateau, but with different parameter bindings.

The following four plateaux, all with identical profiles, share another different action sequence. The final plateau observed in solving the problem occurs once for every problem with an odd number of philosophers[1]; this action sequence is only used once per problem instance but if cached for use on future problems will be reusable. The pattern of plateaux observed here is seen in all the problems, with different numbers of the different types of plateau being encountered. The first two plateau types are seen once during search for each pair of Philosophers in the problem. All of the plateaux occurring in this domain are identifiable as

[1]These are even numbered problems since problem 1 starts with two philosophers.

corresponding to the same lifted action sequence to many others. The properties of this domain suggest that the use of macro-actions should theoretically improve performance as several similar plateaux are encountered. Optimal planning in this domain is in P for the IPC benchmark problems; in the general case it is PSPACE-hard [30].

The Optical Telegraph Domain

The Optical Telegraph domain is the second of the two model-checking-based domains. It is again concerned with checking solutions to problems and identifying any deadlocks that may occur. The heuristic landscape in this domain is similar to that in the Philosophers domain: that is, there are many plateaux which require the same action sequence to escape. It can be seen from the corresponding graph in figure 5.2 that many of the plateaux have identical profiles. There are four distinct plateau types: the first occurs only once at the beginning of the plan; the second and third type alternate in the following stages of the plan, and the final stage of the plan consists of many instances of the fourth type of plateau.

Theoretically the search performance would be greatly improved by using macro-actions in this domain since the identical plateaux could be overcome quickly by the application of a single macro-action generated on the first. As the properties of this domain are similar to those of the philosophers domain other domains have been selected in favour of this one: the performance observed in the two domains would be expected to be similar. The Philosophers domain remains in the evaluation suite to investigate the benefits of using macro-actions in domains with this type of heuristic properties. The complexity of this domain is the same as that in the Philosophers domain [30].

The Satellite Domain

Satellite was the only domain available in a strips formulation, to be used in both IPC 3 and IPC 4. In this domain a number of imaging tasks must be accomplished using several satellites. The heuristic profile shown in figure 5.2 for this domain reveals that very few plateaux are encountered during the search to solve the corresponding problem. The example profile given is typical of problems in the domain; plateaux are rarely encountered, and when they are encountered they are short. The short plateaux mostly correspond to the action sequence turn-to take-image; turn-to switch-on is another short plateau-escaping sequence. Other plateaux do occasionally occur in the problem suite but there are relatively few, and most require only two actions for escape.

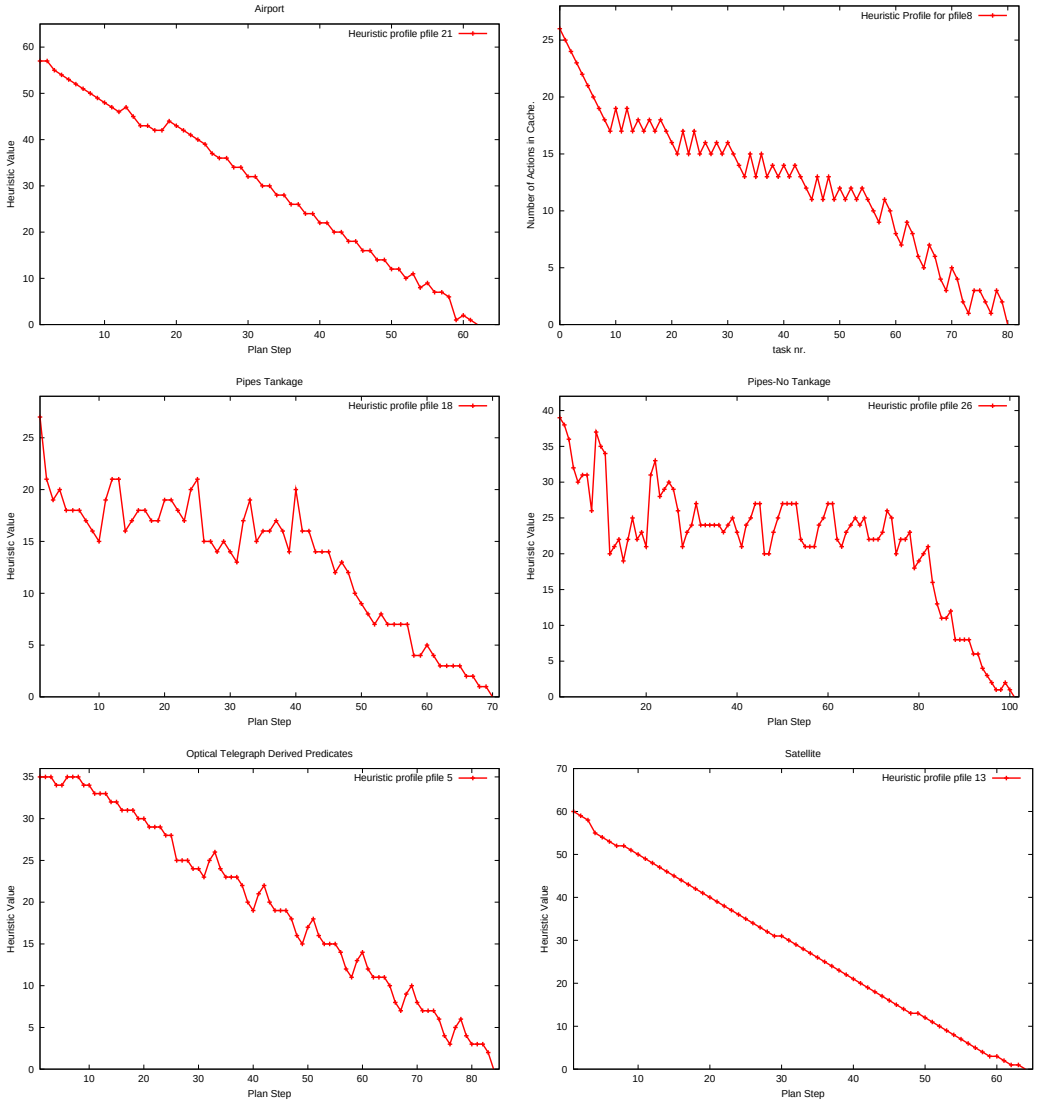


Figure 5.2: Heuristic profile across the solution plan for a problem selected from each of the IPC4 evaluation domains (from left to right: Airport, Philosophers, Pipes Tankage, Pipes No-Tankage, Optical Telegraph (derived predicates) and Satellite).

The results of running tests in this domain will produce an insight into how much overhead the macro-actions cause to the planner and whether it is possible to observe a performance improvement when the plateaux to be escaped are small and infrequently occurring. Planning in this domain is in P, optimal planning in this domain is NP-equivalent [30].

The Pipesworld Domains

This is a domain concerned with modelling the transportation of oil products through pipelines that was introduced in IPC4. The Pipesworld domains have different properties to other transportation problems considered, such as the logistics-style Drivelog and Depots domains, as the pipelines are constantly filled with oil. Pushing an oil product into the start of a pipeline will have the effect of pushing a, potentially different, product out of the other end. The strips version of the domain has two variants: tankgage, which includes the additional restriction that a finite amount of oil can be stored at each location at the end of a pipe; and no-tankage, which ignores this restriction.

The plateaux in the Pipesworld domains can have arbitrarily large exit depths [36]; that is arbitrarily long macro-actions can be required to escape them. Hoffmann [31] remarks that the scaling performance of many planners using the RPG heuristic in this domain is somewhat surprising as long plateau-escaping sequences do occur in the benchmark suite. The potential for long plateau-escaping sequences could have two possible effects on the performance of Marvin. The performance could degrade if there are long plateaux which require very different action sequences to escape, if the lengths can vary dramatically it is less likely that two plateaux will be the same. Conversely it may be the case that, although the lengths of plateau-escaping sequences can vary dramatically, the nature of the problem does in fact give rise to a number of similar plateaux. If the similar plateaux are those that are observed to have long exit depths then the application of macro-actions could potentially improve performance.

The complexity of planning in both of the pipesworld domains is NP-Hard [30]. As the two domains have similar properties only one of them has been selected for use in the evaluation: the No-Tankage version. The No-Tankage version has been selected as the planner can solve more problems in this domain and the search space is undirected so macro-actions are likely to be more successful. The success of macro-actions in directed search spaces will be tested through the inclusion of the Free Cell and Airport domains.

5.1.3 Other Domains

The Briefcase Domain

The Briefcase domain has been chosen for inclusion as an evaluation domain for several reasons. Many macro-action generation systems and planning systems learning other information are evaluated on this domain; it is therefore a useful domain to evaluate on for comparison. Further, it has a structure not similar to many of the other evaluation domains in which very short plateaux occur in reasonably large numbers. The plateaux that occur in this domain are all plateaux of length 2 and occur frequently in solving problems. The success of using macro-actions in this domain will depend on whether overheads of managing macro-actions are overcome by the use of macro-actions to escape plateaux that are very short (i.e. that the planner could escape reasonably quickly by exhaustive search). The heuristic profile shown in figure 5.1 for this domain was generated on a smaller problem than those used in the tests: any larger problems have large makespans and the profile cannot be easily seen in the graph. The profiles in the evaluation suite do, however, have the same structure. The problems used are 20 randomly generated instances with the number of objects being increased, in increments of 10, from 50 to 240.

5.1.4 Summary of Selected Domains

Table 5.1 shows the final choice of domains selected for evaluation; along with a brief summary of the structure of the search space, under the RPG heuristic, in these domains. The domains selected have a diverse range of properties under the RPG heuristic. Further, they have a range of complexities from the PSPACE-hard Airport problems and the NP-hard Pipes No-Tankage domain, to domains such as Driverlog for which planning is in P. It is worth noting that, although planning for some domains is in P, the problems still remain hard to planners: one of the most difficult domains for planners in IPC 4 was the PSR domain, which is in fact in P[2]. Overall the selected domains should give a good insight in to the effects of using plateau-escaping macro-actions in planning as many different problem structures have been considered.

Where available the ADL version of each domain is used, if not the typed STRIPS version of the domain is used. Marvin does not deal with temporal or metric planning domains.

[2]Malte Helmert: Presentation on "New Complexity Results for Classical Planning Benchmarks", ICAPS 2006.

Domain	Structure of Search Space Under RPG Heuristic
FreeCell	Directed search space with varied structure and many complex plateaux.
Airport	Small plateaux but many unpredicted inescapable dead ends, directed search space.
Depots	Varied: some problems have few plateaux all of which are small; others have many large complex plateaux.
Driverlog	Variation of plateaux within problems, some plateaux occur with reasonable frequency; others occur only once.
Philosophers	Regular structure repeated long plateaux.
Pipes NT	Large plateaux with arbitrary exit depths.
Briefcase	Frequent occurrence of short plateaux.
Satellite	Short infrequently occurring plateaux.

Table 5.1: Summary of Chosen Evaluation Domains

5.1.5 Experimental Methodology

Experiments have been carried out in order to evaluate several hypotheses. The hypothesis being tested is given at the beginning of each section; followed by a detailed analysis on a per-domain basis; and finally a summary of the conclusions drawn from each set of experiments. Results are shown for all of the evaluation domains wherever possible. All experiments have been done across 30 identical computers each of which has 1GB of RAM and a 3.4 GHz Intel Pentium 4 processor. Each run of the planner is subjected to time and memory limitations: if the planner does not solve the problem within 30 minutes or uses more than 800MB of memory it is deemed to have failed to solve the problem. The time limit chosen is the same as that used in IPC 4; the memory limit used is slightly lower than the 1GB limit used in IPC 4 due to practical limitations on the machines used.

Performing a full cross evaluation would not be possible: the results presented in this chapter correspond to over 2500 CPU hours (over 110 CPU days). Evaluating each of the caching strategies with different numbers of macro-actions before and after, and all of the combinations of caching strategies, would be infeasible. Instead the experiments have been done individually with one parameter changed in each test and compared to a standard control version. This allows the effect of each feature to be isolated. The standard control version for the macro-action tests (referred to as no macro-actions) is as follows.

- No macro-action generation, use or caching;
- Concurrency enabled;
- Symmetric action pruning enabled;
- Using the RPG heuristic.

In the case of the Causal Graph heuristic tests the control version is configured in this way but uses the Casual Graph heuristic rather than the RPG heuristic. For the action reordering evaluation a different control is used, configured as the above RPG control but with concurrent planning disabled to ensure that it is clear which action is the last action in the plan to update.

Where appropriate other versions of the planner have been shown for comparison, for example the no macro-action caching version of the planner in the macro-action caching tests. This version is identical to the control in all features except that generation and use of macro-actions is enabled (note that caching is, of course, still disabled). In all versions using macro-actions the planner is applying all macro-actions after other actions unless otherwise stated. The no caching version of Marvin is an important version for comparison: this is a planner that has been internationally benchmarked against many other planning systems in the Fourth International Planning Competition [31]. This is therefore used as a benchmark to test the performance of turning on and off certain features of the planner wherever possible. Comparison to other planners can be made by comparing the results given here to the performance of that version, both in this thesis and in the competition results [31].

Tables quoting mean makespan improvements show the mean of the makespan of plans generated by the control version of the planner minus the makespan of the plan by each of the versions of the planner to be tested on the same problem. That is, $\sum_{i=0..n} (Makespan(C_i) - Makespan(V_i))$, where n is the number of problems in the domain and Makespan(C_i) and Makespan(V_i) represent the makespan of the plans generated by the control and the other version of the planner respectively. Only problems solved by both the individual configuration and the no macro-actions version of the planner are included in calculating these results; the figures in brackets indicate the number of mutually solved problems. The mean time improvement tables present the time taken to solve problems in the same manner.

Pre-experimentation has been done in order to select an appropriate range of values to use for each parameter that best demonstrates the effect of changing

that parameter. This consisted of running a small number of experiments to determine where the interesting changes are likely to occur within the range for each parameter to allow the larger scale experiments to focus on these more interesting areas. It is, of course, not possible to test all values of all the parameters: for example, applying 1 macro-action before other actions then incrementing by 1 until the maximum library size has been reached would require an infeasible amount of time. The values chosen for the parameters, therefore, give a representative cross-section of the effect of changing the parameter in question on performance. In the interests of clarity and conciseness not all the versions tested are shown in the graphs of results. Other versions are, however, included in tables or discussed in the text as appropriate.

5.1.6 Statistical Testing

Significance tests have been carried out in order to verify the stated hypotheses. The test used to determine significance is the Wilcoxon signed-rank test. This test was selected due to the nature of the data generated: the test does not rely on assumptions about the data being normally distributed, which in this case is not true. The null hypothesis for the tests is that the versions of the planner being compared perform identically. All tests performed attempt to refute this hypothesis to a level of 95% confidence unless otherwise stated. The tests that do not use 95% confidence use a higher confidence level in order to show a total ordering on the performance of the planner: to impose the ordering A is better than B is better than C to 95% confidence it is necessary to demonstrate that A is better than B, and B is better than C both with probability greater than $\sqrt{0}.95$.

The tests are carried out on the pairwise values of time taken to solve each problem by each of the versions of the planner to be considered; results across all domains are considered in one test. The procedure used to perform the test is as follows. First, the differences between the pairs of values (time taken by configuration A minus time taken by configuration B) are calculated for all mutually solved problems. Pairs of values with difference equal to zero are ignored. The remaining pairs of values are then ranked according to the magnitude of their differences, i.e. the signs are ignored. The pair with the highest valued difference is given the highest numbered rank; that with the smallest difference is given a rank of 1. If any pairs have differences of equal magnitude, they are each given the same ranking equal to their mean rank; e.g. if two equal pairs would have ranks 1 and 2 then they are both given rank 1.5. The test requires three values

to be calculated: the number of pairs (N), the sum of all the positive rankings (W+) and the sum of all the negative rankings (W-). To compute the latter two of these the signs are reintroduced accordingly after ranking. These values are then substituted into the standard formula to compute the probability that the null hypothesis is correct[3].

5.2 Planning With Macro-Actions

This section addresses some of the issues encountered when planning with plateau-escaping macro-actions. The versions of the planner considered here are the no macro-actions control version of the planner; one version of the planner generating macro-actions on a per-problem basis (no caching); and another keeping all macro-actions ever generated in solving the problems so far. This section is only concerned with the effects of macro-actions in these three configurations.

The first issue is a commonly discussed phenomenon regarding the makespan of plans generated using macro-actions. It is often a concern that although using macro-actions will improve the speed with which a plan can be generated the makespan may be adversely affected. The second issue is whether to apply plateau-escaping macro-actions before, or after all other actions during search. Applying macro-actions after other actions corresponds to only applying such actions when the planner has reached a plateau and minimising their impact on normal search. Applying them before gives rise to potential maximum destructive effect but also the maximum speed up if the macro-actions are frequently the correct actions to apply. The study performed in this section is an analysis of the length of the useful macro-actions generated.

5.2.1 The Effect of Macro-Actions on Makespan

It is clear that in strategies like that used by Korf [43] the macro-actions will most likely degrade solution quality. Achieving one goal then maintaining it whilst achieving others is often not the most efficient way of solving problems. In other strategies it is not as clear why selecting a macro-action would necessarily increase the makespan of plans. If the planner can find a more optimal route off a small plateau (there are fewer possible routes so finding the optimal route is more probable) and then re-use this in a larger problem, where the optimal route may

[3]Instead of computing the probability it is possible to look up values in a table, however, here the computation was done using the formula due to unavailability of a table with data for sufficiently large values of N.

Domain	No Caching	Keep All
FreeCell	3 *(18)*	8.56 *(16)*
Airport	0 *(38)*	0.5 *(38)*
Depots	1.62 *(13)*	3.5 *(14)*
Philosophers	0 *(48)*	0 *(48)*
Driverlog	-7.88 *(17)*	-0.16 *(17)*
Pipes NT	-2.71 *(38)*	-2.41 *(39)*
Briefcase	120.93 *(15)*	122.4 *(15)*
Satellite	3.25 *(36)*	5.86 *(36)*
Totals	**14.78** ***(223)***	**17.28** ***(223)***

Table 5.2: Mean of makespan of the solution plan generated by the no macro-actions version minus makespan of plan generated using macro-actions. Results are calculated on mutually solved problems.

be less clear, the makespan of plans may indeed decrease. Note that Marvin uses best-first, not breadth-first, search on plateaux [14] so plateau-escaping sequences are not guaranteed to be optimal.

Hypothesis

The use of plateau-escaping macro-actions in planning does not increase the makespan of solution plans.

Analysis

Table 5.2 shows a summary of the makespan of plans generated of running the planner on the problems in the evaluation domains. A graphical presentation of the full results is given in figure 5.3. The effects of macro-actions on makespan when using the various macro-action library pruning strategies will be considered in the corresponding sections.

The macro-actions in the FreeCell domain cause the planner to take a different route through the EHC search space and generally to find a plan which is

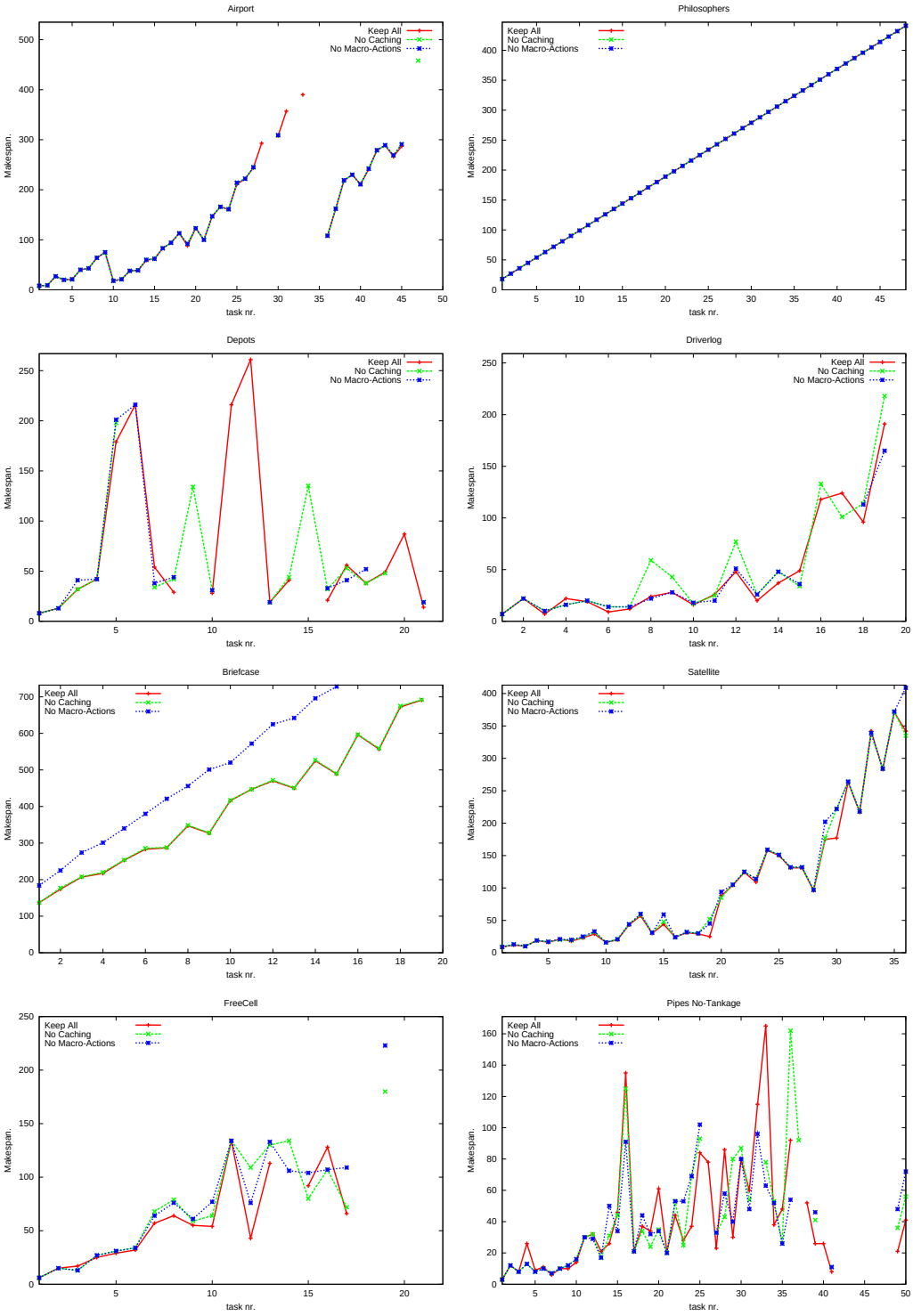


Figure 5.3: Makespan of plans generated with and without macro-actions across the evaluation domains (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

nearer to optimal. It is, however, possible to observe in figure 5.3 that in some problems the route taken is further from optimal; overall, the macro-action versions of the planner do generally generate shorter plans. In the Airport domain many problems are solved by best-first search, in which the macro-actions are not as useful as they are in EHC. The slight makespan improvement that arises in the keep all version is a result of macro-actions allowing the planner to solve problems by EHC on which the no macro-actions version had to resort to best-first search. When the planner resorts to best first search it is too expensive to reason about concurrency. Plans generated by Marvin using EHC are therefore generally shorter than those generated by best-first search. There is not a great deal of scope for the introduction of concurrency in the Airport domain, and the number of large problems the control successfully solves by best-first search is small, so the improvement is only slight.

The impact in the Depots domain is slightly larger, again the macro-actions allow the planner to solve problems without resorting to best-first search. The potential for concurrency is, however, greater in the Depots domain, and more problems are solved by EHC rather than best-first search so more improvement is seen. Discounting those problems not solved by both configurations using EHC the no caching version still maintains a 0.43 step advantage and the keep all version a 2.29 steps, the macro-actions are guiding EHC off plateaux in a more efficient direction.

The Driverlog and Pipes No-Tankage domains are the only two domains in which the makespan is increased by the use of macro-actions. In the Driverlog domain best-first search actually, in general, makes shorter plans than EHC because the planner makes bad decisions when solving the problem using EHC. The macro-actions generated by the two macro-action versions help the planner to solve more problems by EHC in Driverlog, without the need to resort to best-first search. Unfortunately this does not necessarily lead to the best solutions. On problems mutually solved by the keep all and control versions using EHC the plans generated by the keep all version are actually, on average, 1.33 steps shorter. The Pipes No-Tankage domain is the only domain in which the plans in generated both in general and on problems mutually solved by EHC are longer for both macro-action versions. In this domain the plateaux can have very long exit depths and some macro-actions that were generated on long plateaux are also reusable on shorter plateaux. This is a phenomenon that will, of course, occur in many domains but only occurs frequently enough to cause an overall disadvantage in this domain where plateaux can be very large. The increase in makespans of

	No Caching	No Macro-Actions
Keep All		
(p =)	$8.978 * 10^{-06}$	$2.743 * 10^{-07}$
Sig?	Yes	Yes
Best	Keep All	Keep All
No Caching		
(p =)		0.001858
Sig?		Yes
Best		No Caching

Table 5.3: Significance table for Makespan Improvement When Using Macro-Actions using the Wilcoxon signed-rank test: p is the probability that the null hypothesis, that the versions produce plans of equal length, cannot be rejected; sig? denotes whether or not the null hypothesis can be rejected with probability > 0.975; Best is the best performing of the two configurations being compared

plans is, however, only slight with approximately two steps of increase for both versions.

In the Philosophers domain all plans generated are optimal: there is only one collection of actions that will solve the problem [30] (although the ordering of these actions can be varied to a certain extent). The macro-actions do, however, allow the planner to find the same plan in a greatly reduced length of time. The makespan improvement in the Briefcase domain is a result of macro-actions that force the planner to consider loading or unloading an object into a briefcase when moving it; rather than uselessly moving the briefcase between locations without doing anything (this is necessarily redundant as the location map is fully connected). This is an example of information learnt on smaller plateaux that is then used again when larger problems are encountered. A similar phenomenon occurs in the Satellite domain: this time the redundant actions are turn_to actions: Satellites can turn to any phenomenon from any phenomenon, yet a weakness in the heuristic causes the planner to insert redundant sequences of turn_to actions into the plan. All successfully solved problems in both the Satellite and Briefcase domains are solved using EHC by all versions.

Conclusion

It can be seen that in many domains the macro-actions actually allow the planner to generate shorter plans than the control version. Overall the no caching version manages to find plans that are, on average, almost 15 steps shorter than the no macro-actions version; whilst the keep all version gains an improvement

of 17 steps. Table 5.3 shows that the makespan improvements have been shown to be significant using a Wilcoxon signed-rank test. The confidence level chosen for comparisons in this experiment is 97.5% in order to allow a total ordering to be placed on the versions with confidence 95%. The version generating macro-actions on a per-problem basis (no caching) generates plans with significantly shorter makespans than the no macro-actions version. Further the keep all version, caching macro-actions, offers significantly better performance than the no caching version. Indeed a total ordering on these versions of keep all being better than no caching in turn being better than no macro-actions has been shown to 95% confidence. These tests support the proposed hypothesis that plateau-escaping macro-actions do not increase the makespan of plans; furthermore they show a stronger conclusion: that plateau-escaping macro-actions can decrease the length of solution plans.

Learning macro-actions on smaller plateaux can enable the planner to find a more optimal route off larger plateaux. Smaller plateaux have fewer suboptimal escape routes; there are therefore fewer possible other irrelevant actions that may be applied on the way to finding an escape from a plateau. On large plateaux there are many entities and many possible actions that could be added to the escape sequence unnecessarily. Another factor is that macro-actions can allow the planner to solve problems via EHC, a framework in which it is feasible to reason about concurrency, rather than best first search, thus allowing shorter plans to be generated. This phenomenon is, however, not solely responsible for the makespan improvement in plans as many plans generated by both configurations using EHC only are shorter in versions using macro-actions.

Macro-actions can be applied on plateaux where a shorter exit path could otherwise have been found making the makespan of some problems longer. This phenomenon does not, however, occur so frequently as to outweigh the other benefits gained.

5.2.2 The Order in which to Present Macro-Actions to the Planner

In its default configuration, Marvin consideres macro-actions after all other actions when planning. In EHC search, this results in macro-actions only being considered when a plateau is encountered: if a strictly better single action successor existed it would have been discovered in considering each of the single actions and added to the plan accordingly. When all single actions are considered and no better state results from any of these then search has reached a

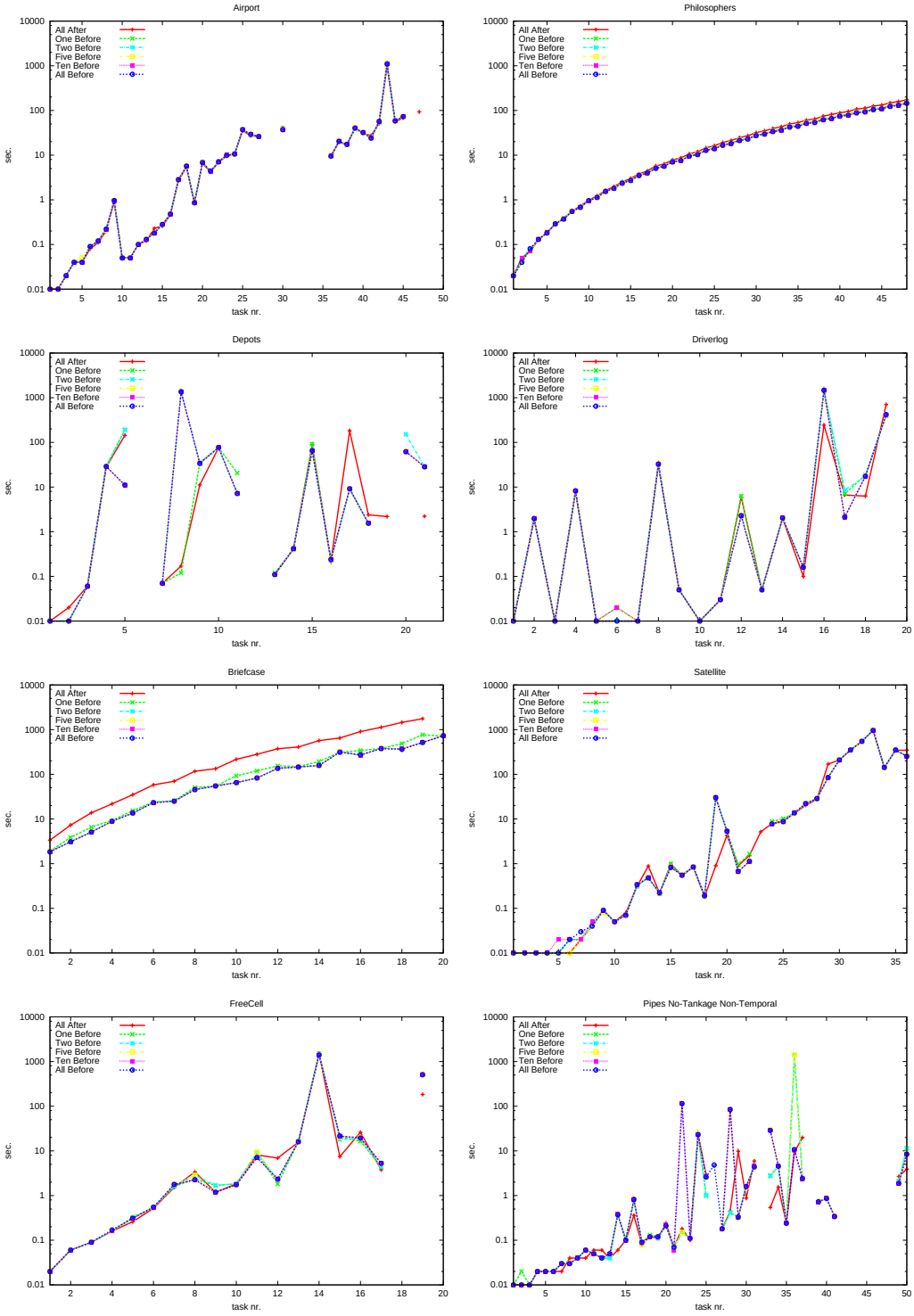


Figure 5.4: Time Taken To Solve Problems Applying Differing Numbers of Macro-Actions Before Unit Actions with No Macro-Action Caching (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

plateau so the macro-actions will be considered, since they follow the other actions in the list. It may be the case that considering macro-actions after other actions is not the most efficient strategy. It is therefore prudent to investigate the effects of considering macro-actions before other actions.

There are a number of possibilities, the first is to consider all macro-actions before all other actions; this may be inefficient if many unhelpful macro-actions are generated. Another approach is to consider a collection of macro-actions before unit actions, then the unit actions, and finally the remaining macro-actions. Two sets of experiments have been performed to show the effects of applying macro-actions before other actions. The first does not allow macro-action caching, offering a fairer comparison between versions as different macro-action libraries cannot be generated. The second allows macro-action caching as the effects of applying macro-actions before all other actions will become much more apparent when more macro-actions are present.

Hypothesis

The experiments in this section investigate two hypotheses:

- Applying macro-actions before all other actions in the search space will cause the performance of the planner to degrade. Considering macro-actions after all other actions, i.e. only when plateaux are encountered, will offer search guidance when the heuristic is weak without the performance suffering as much.
- In the case where the planner has a large collection of macro-actions the detrimental effect on performance will be more dramatic than in cases where the planner has fewer macro-actions.

Analysis of Data With No Macro-Action Caching

The results of running the planner without doing macro-action caching and considering differing numbers of macro-actions before unit actions are shown in figure 5.4. The number of macro-actions to be applied before all other actions indicates the maximum number of macro-actions that can be considered before other actions. All configurations begin search with no macro-actions to consider: none have yet been generated.

Domain	All After	1 Bef.	2 Bef.	3 Bef.	5 Bef.	10 Bef.	All Bef.
FreeCell	18	18	18	18	18	18	18
Airport	39	38	38	38	38	38	38
Depots	17	17	18	18	18	18	18
Philosophers	48	48	48	48	48	48	48
Driverlog	19	19	19	19	19	19	19
Pipes NT	39	39	40	40	40	40	41
Briefcase	19	20	20	20	20	20	20
Satellite	36	35	35	35	35	35	35
Totals	**235**	**234**	**236**	**236**	**236**	**236**	**236**

Table 5.4: Coverage Across Evaluation Domains Applying Macro-Actions Before Other Actions with no Macro-Action Caching

When the first plateau is escaped there is now one possible macro-action to consider before other actions; any configuration considering more than one macro-action before other actions will consider this action first. At this early stage, since there is only one macro-action the configuration of the planner applying ten macro-actions before all other actions will behave identically to the configuration of the planner applying one macro-action before all other actions.

As more plateaux are encountered during search, more macro-actions are generated and the different configurations of the planner begin to exhibit different behaviour, with differing numbers of this collection of macro-actions being applied before the unit actions. When not caching the macro-actions the benefits of not considering large numbers of macro-actions before the unit actions are not as apparent since it is rare that large numbers of macro-actions are generated. Further if large collections of macro-actions are generated they are only present during the later stages of search. The relative performance of all the strategies is similar in this case with the planner; it remains interesting to compare the performance of the strategies in this case, however, since the performance of the planner is only being affected by the change in ordering strategy, rather than by the collection of macro-actions it has generated.

Table 5.4 shows that there is little difference in the number of problems solved overall by each configuration with the versions applying more than one macro-action before other actions solving just one more problem than the standard, all after, configuration. There is also little difference in the speed with which problems are solved, shown in figure 5.4: some problems are solved more quickly by the version applying all macro-actions after other actions (Satellite problem 19, for example); some being solved more slowly (e.g. Depots problem 17).

Domain	**1 Bef.**	**2 Bef.**	**3 Bef.**	**5 Bef.**	**10 Bef.**	**all Bef.**
FreeCell	-17.08 *(18)*	-16.21 *(18)*	-14.09 *(18)*	-12.76 *(18)*	-11.27 *(18)*	-11.46 *(18)*
Airport	-2.64 *(38)*	-2.80 *(38)*	-2.91 *(38)*	-2.80 *(38)*	-3.34 *(38)*	-2.84 *(38)*
Depots	4.34 *(16)*	-78.88 *(16)*	-78.8 *(16)*	-67.5 *(16)*	-67.41 *(16)*	-67.64 *(16)*
Philosophers	5.12 *(48)*	5.68 *(48)*	6.41 *(48)*	6.43 *(48)*	6.41 *(48)*	6.42 *(48)*
Driverlog	-49.39 *(19)*	-49.13 *(19)*	-50.26 *(19)*	-49.26 *(19)*	-49.47 *(19)*	-49.52 *(19)*
Pipes NT	-35.56 *(39)*	-35.56 *(39)*	-36.04 *(39)*	-38.26 *(39)*	-5.30 *(39)*	-5.30 *(39)*
Briefcase	263.72 *(19)*	293.34 *(19)*	293.96 *(19)*	293.34 *(19)*	293.34 *(19)*	293.89 *(19)*
Satellite	3.98 *(35)*	3.99 *(35)*	3.89 *(35)*	3.87 *(35)*	4.07 *(35)*	3.91 *(35)*
Totals	**21.56** ***(232)***	**15.05** ***(232)***	**15.26** ***(232)***	**16.63** ***(232)***	**20.88** ***(232)***	**20.93** ***(232)***

Table 5.5: Mean of time taken by version considering all macro-actions after other actions minus time taken using each other version on mutually solved problems

The two domains that show a consistent improvement by applying all macro-actions before are Philosophers, with a slight improvement, and Briefcase with a more significant improvement. Indeed an overall time improvement is seen in table 5.5 as a result of the great performance in the Briefcase skewing the mean upwards. These two domains are, however, both highly structured and macro-actions are always the correct thing to apply when they are applicable; this is why the performance is improved by considering the actions first. The other domains do not share this property, the performance shown in table 5.5 is consistently worse.

	1 Before	**2 Before**	**3 Before**	**5 Before**	**10 Before**	**All Before**
All After (p =)	0.008236	0.001159	0.0008645	0.000431	0.0003891	0.0002871
Sig?	Yes	Yes	Yes	Yes	Yes	Yes
Best	1 Before	2 Before	3 Before	5 Before	10 Before	All Before
1 Bef. (p =)		0.03227	$2.204 * 10^{-05}$	0.0001415	$1.091 * 10^{-05}$	$6.815 * 10^{-06}$
Sig?		Yes	Yes	Yes	Yes	Yes
Best		2 Before	3 Before	5 Before	10 Before	all Before
2 Bef. (p =)			0.0008415	0.0904	0.001222	0.0002285
Sig?			Yes	No	Yes	Yes
Best			3 Before		10 Before	All Before
3 Bef. (p =)				0.04171	0.4693	0.9275
Sig?				Yes	No	No
Best				3 Before		
5 Bef. (p =)					0.3407	0.03258
Sig?					No	Yes
Best						All Before
10 Bef. (p =)						0.2417
Sig?						No
Best						

Table 5.6: Significance table applying different numbers of macro-actions before other actions: p is the probability that the null hypothesis cannot be rejected; sig? denotes whether or not the null hypothesis, that the versions take the same length of time to solve problems, can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

Conclusion for No-Caching Version

The combination of the small numbers of macro-actions present when not caching macro-actions, and search-time helpful action pruning lead to the result that applying macro-actions before other actions appears to give some performance gains in domains where macro-actions are useful. This is achieved without causing a great degradation in performance in domains where the macro-actions are not useful. Table 5.6 shows that there is, in fact, significant improvement when applying macro-actions before other actions. This is, however, a result of the good performance in the Briefcase domain: if the significance tests are performed excluding this domain no version of the planner can be shown to be significantly better than the all after version. The gains are not allowing the planner to solve significantly more problems and when more macro-actions are present (due to macro-action caching) the overheads introduced are likely to increase.

Analysis of Data Collected when Keeping all Macro-Actions

Figure 5.5 shows the results of running Marvin on the evaluation domains applying different numbers of macro-actions before the other unit actions. In order to ensure that there will be sufficient macro-actions for the planner to consider the macro-actions are cached between problems; no pruning strategy is employed so the library contains all macro-actions ever generated by the planner. The macro-actions are ranked according to the number of times they have been used, so when applying n macro-actions before other macro-actions the n most used macro-actions will be the ones considered first in the order most used to least used (ties are broken by using the oldest macro-actions first and then arbitrarily). These results must be considered in conjunction with the results from the tests with no caching because different orderings of macro-actions, with respect to unit actions, will cause search to proceed in different directions and experience different plateaux. Experiencing different plateaux will result in different libraries of macro-actions being built for the different configurations. The two sets of results when both considered can, however, give an insight into the best ordering policy to select.

The version considering all macro-actions after unit actions (i.e. only considering macro-actions on plateaux) is shown as a control. In the Philosophers domain, where macro-actions provide excellent search guidance performance is improved by considering macro-actions before other actions, this is because when the macro-actions are applicable, and helpful, they are the correct action to apply, hence considering them first, rather than exhausting the other possibilities allows

Domain	All After	1 Bef.	2 Bef.	5 Bef.	10 Bef.	All Bef.
FreeCell	16	17	17	19	17	17
Airport	41	39	40	38	38	40
Depots	19	21	19	15	16	17
Philosophers	48	48	48	48	48	48
Driverlog	19	19	19	16	19	19
pipes NT	42	41	38	42	40	41
Briefcase	19	17	20	20	20	20
Satellite	36	31	32	31	27	23
Totals	**240**	**233**	**233**	**229**	**225**	**225**

Table 5.7: Coverage Across Evaluation Domains Considering Macro-Actions Before Other Actions Caching All Macro-Actions

the planner to solve problems more quickly. After the number of macro-actions to be considered before other actions has reached a certain value the performance remains the same. This is because all of the useful macro-actions are already being considered before other actions and in this domain, under EHC search, only finite number of different plateaux occur so no more useless macro-actions will be generated. The Briefcase domain shows similar behaviour with a library of two macro-actions, which are again the correct actions to apply. Overall the results from the remaining domains show that considering two or more macro-actions before the other actions degrades planner performance considerably. Applying one macro-action before all other actions decreases coverage in most domains, but in some domains coverage is slightly improved.

In the Airport domain the number of macro-actions considered before or after other actions has only a slight impact on performance. Table 5.7 shows that the version of the planner using the default (all macro-actions after) configuration solves the most problems, 41 out of the 50 problems. The version applying all macro-actions before other actions has the next best coverage solving 40 of the problems. Of the problems solved by both the control and each individual configuration the version of the planner applying one macro-action before the other actions solves problems on average the fastest; with the version applying two macro-actions before the other actions being slightly faster than the control. All other configurations both solve fewer problems and solve problems slower, on average, than the default configuration.

In the Depots domain, the version considering all macro-actions after the unit actions solves 19 of the problems; the only version to solve more problems is the version considering only one macro-action before other actions. The other

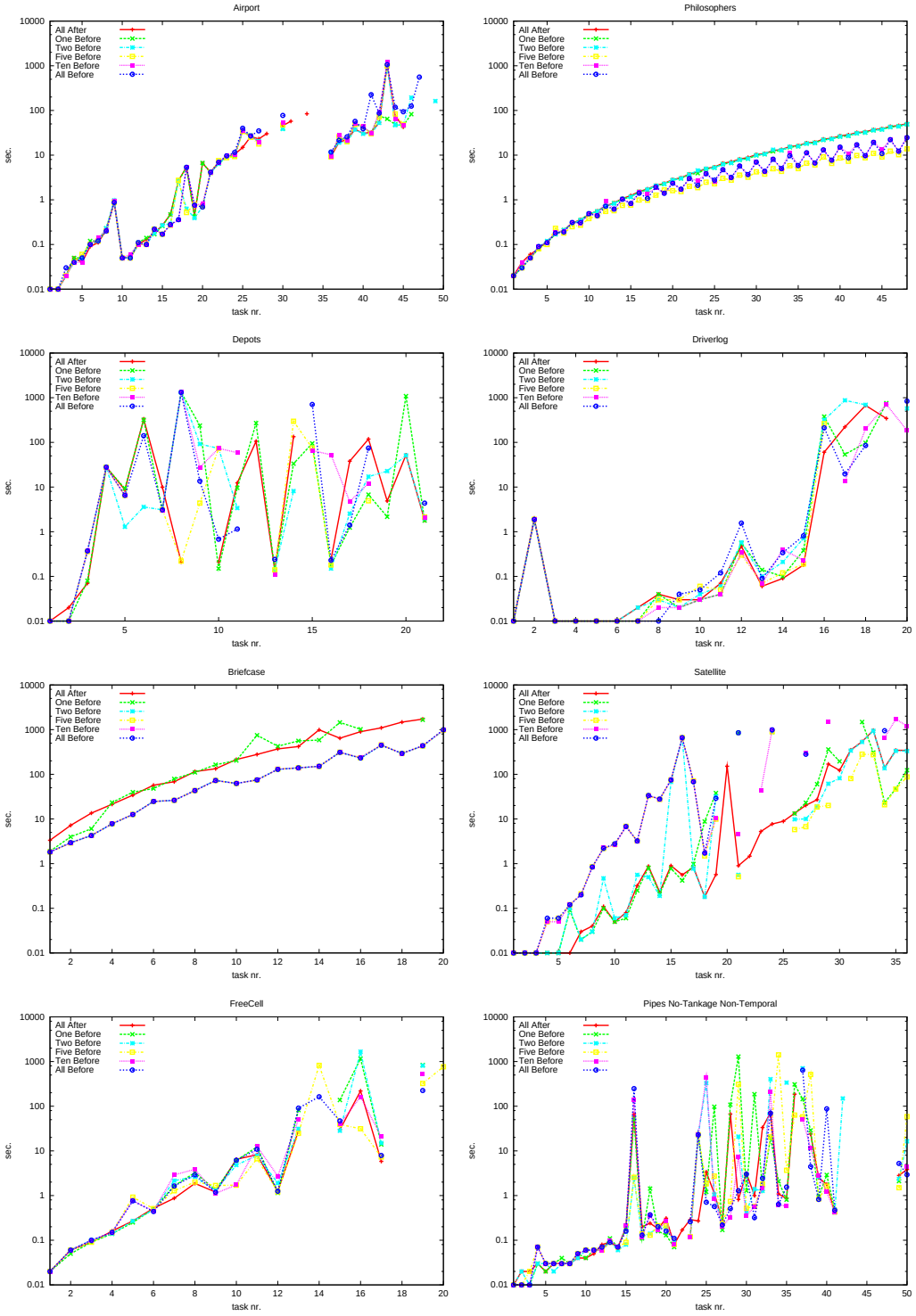


Figure 5.5: Time taken to solve problems applying different numbers of macro-actions before unit actions, caching all macro-actions (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	**1 Bef.**	**2 Bef.**	**5 Bef.**	**10 Bef.**	**All Bef.**
FreeCell	-69.96 *(16)*	-90.64 *(16)*	11.6 *(16)*	0.30 *(16)*	-5.73 *(15)*
Airport	24.38 *(38)*	1.77 *(38)*	-0.68 *(38)*	-7.7 *(38)*	-11.89 *(38)*
Depots	-118.04 *(19)*	-44.25 *(18)*	-8.69 *(13)*	-96 *(14)*	-68.63 *(15)*
Philosophers	0.42 *(48)*	0.39 *(48)*	8.11 *(48)*	6.26 *(48)*	6.41 *(48)*
Driverlog	0.59 *(19)*	-51.78 *(18)*	-12.61 *(16)*	17.51 *(18)*	35.28 *(18)*
Pipes NT	-41.56 *(40)*	-26.39 *(36)*	-47.48 *(41)*	-14.40 *(39)*	-4.18 *(40)*
Briefcase	-66.53 *(17)*	321.12 *(19)*	320.32 *(19)*	320.32 *(19)*	320.64 *(19)*
Satellite	-55.51 *(31)*	-43.03 *(32)*	9.17 *(31)*	-232.62 *(27)*	-165.39 *(23)*
Totals	**-40.76** ***(228)***	**8.4** ***(225)***	**34.96** ***(222)***	**-0.78** ***(219)***	**13.30** ***(216)***

Table 5.8: Mean of time taken by version considering macro-actions after other actions minus time taken considering varying numbers of macro-actions before other actions, caching all macro-actions. Results are calculated on mutually solved problems.

versions all solve fewer problems as they are forced to consider many macro-actions that are not useful in solving the problem, before getting to the other unit actions. The coverage decreases as the number of macro-actions applied before increases until the version considering ten macro-actions before is reached. At this point the number coverage begins to increase slightly, but not to the levels it was at with first configurations. The increase in coverage from 15 to 16, for the configurations using five and ten macro-actions before other actions respectively, is a result of problem file 11 being solvable using only EHC with the set and ordering of macro-actions the version considering ten macro-actions before other actions is using; the version applying five macro-actions before resorts to best-first search and subsequently fails to solve the problem within the alloted time. The further increase in coverage for the version considering all macro-actions before is a result of the same phenomenon on a different problem instance; again the coverage achieved by this configuration is still not as good as that achieved by the version considering all macro-actions after other actions. On problems solved by

each configuration and the control (using all macro-actions after unit actions) the control solves problems faster on average than all configurations. The one before configuration that solved more problems solves mutually solveable problems, on average, 118 seconds more slowly.

When solving problems in the Driverlog domain all versions of the planner have similar coverage. None of the versions applying macro-actions before other actions solve more problems than the version applying all macro-actions after other actions. The versions applying 10 and all macro-actions before do solve mutually solved problems more quickly on average, as shown in table 5.8. Examining the search trajectory taken by the planner, however, it is not the number of macro-actions to be applied before other actions that allows for the performance improvement. It is simply the trajectory these versions happen to take through the search space as a result of selecting a macro-action that would not otherwise be selected, and the consequent library of macro-actions learnt, that allows for better performance; rather than a direct result of the act of applying the selected number of macro-actions before other actions.

In the Satellite domain it can clearly be seen, in figure 5.5, that the performance deteriorates significantly when macro-actions are considered before unit actions. The configuration considering all macro-actions after other actions is the only configuration able to solve all 36 of the competition instances. This is a domain in which few plateaux occur furthermore, when plateaux do occur they are short. Plateaux escaping macro-actions can, therefore, in theory help the planner in these short plateaux and give rise to a performance improvement. The other consequence of this structure is that there are long sequences of the plan for which a strictly better state can quickly be found using the heuristic as a guide. If macro-actions are considered at all times, instead of simply when the search has reached a plateau, then the performance of the planner can be adversely affected. This phenomenon is also visible in the makespan results in table 5.9 where the macro-actions are being added to the plan when unit actions would have normally been used instead. The configurations considering a small number of macro-actions before other actions do not suffer as dramatic a degradation in coverage as those considering larger numbers of macro-actions before other actions. The only anomaly in the trend is the unexpected performance of the version using ten macro-actions before other actions: examination of the logs shows again that the collection of macro-actions it has discovered allow it to solve the problem using EHC and avoid resorting to best-first search on some of the problems. Its performance is, however, still inferior to that of the default version.

Domain	**1 Bef.**	**2 Bef.**	**5 Bef.**	**10 Bef.**	**All Bef.**
FreeCell	-6.56 *(16)*	-11.36 *(16)*	2.94 *(16)*	-9.56 *(16)*	-11.4 *(15)*
Airport	0.08 *(38)*	1.08 *(38)*	1.32 *(38)*	0.87 *(38)*	-2.26 *(38)*
Depots	6.42 *(19)*	21.5 *(18)*	-3.83 *(13)*	8.36 *(14)*	-1.13 *(15)*
Philosophers	0 *(48)*	0 *(48)*	0 *(48)*	0 *(48)*	0 *(48)*
Driverlog	0.95 *(19)*	-6.15 *(18)*	-1.92 *(16)*	-2.15 *(18)*	8.44 *(18)*
Pipes NT	3.4 *(40)*	5.25 *(36)*	0.68 *(41)*	5.18 *(39)*	1 *(40)*
Briefcase	7.41 *(17)*	83.68 *(19)*	83.68 *(19)*	83.68 *(19)*	83.68 *(19)*
Satellite	10.65 *(31)*	5.84 *(32)*	29.26 *(31)*	22.04 *(27)*	4.7 *(23)*
Totals	**2.79** ***(228)***	**12.48** ***(225)***	**14.01** ***(222)***	**13.55** ***(219)***	**10.38** ***(216)***

Table 5.9: Mean of makespan of solution found by version considering macro-actions after other actions minus makespan of plan generated considering varying numbers of macro-actions before other actions, caching all macro-actions. Results are calculated on mutually solved problems.

Using macro-actions in the FreeCell domain leads to results that are somewhat unpredictable. The macro-actions, and the order in which they are considered relative to other actions, cause the planner to take a dramatically different path through the search space during EHC. Sometimes this path is a better path; and sometimes worse, there is no clear pattern to when the behaviour will be better or worse. This is reflected in the results with the apparently sporadic behaviour, increasing and decreasing coverage and performance, when different numbers of macro-actions are applied before other actions during search. There is no clear pattern of increasing to optimal performance and decreasing either side of this point. It is therefore not possible to make conclusions about the order in which it is best to present macro-actions to the planner in this domain: when different problems are used the best configuration changes.

In the Pipes No-Tankage domain the performance of the planner on mutually solved problems is worse than that of the all after version for all configurations. None of the other configurations solve more problems than the all after version

and only one solves as many problems as this version. In this domain it is the increased number of macro-actions that must be considered before the required action is found that causes the planner to experience larger plateaux. This forces the planner to expand more nodes before finding each strictly better successor during normal search.

Conclusion when Keeping all Macro-Actions

Overall the version considering macro-actions after other actions has solved a total of 240 problems. The total number of problems solved across all domains decreases monotonically, in this experiment, as the number of macro-actions applied before other unit actions increases. Some of the versions of the planner show an overall time improvement but this is due to the results in the Briefcase domain skewing the mean: in most domains the performance on mutually solved problems is made worse by considering any number of macro-actions before other actions. Indeed excluding the Briefcase domain from the results all versions have worse overall performance than the control version as well as lower coverage. The skewed mean, caused by the Briefcase domain, also means that significance tests are unable to show significant difference between most of the versions of the planner in terms of time taken to solve mutually solved problems. The major benefit of applying macro-actions after other actions is in terms of coverage. When the coverage is decreased there are fewer mutually solved problems which makes it more difficult to prove significant time improvement. It is therefore not possible to make a firm conclusion regarding the hypotheses based on the statistical tests considering time taken.

In terms of solution quality table 5.9 shows that some makespan improvement is seen in the Satellite domain, and a great deal of makespan improvement is seen in the Briefcase domain by applying some macro-actions before other actions. This is, however, at the cost of worse makespans in the FreeCell domain and much decreased overall coverage. The default configuration remains throughout the rest of the experiments: when macro-actions are applied during search they are considered after all other unit actions. This corresponds to considering macro-actions only when a plateau is reached in the search space.

5.2.3 Length of Useful Macro-Actions

Many systems using macro-actions impose arbitrary limits on both the number of macro-actions to be used by the system at any given time and the maximum

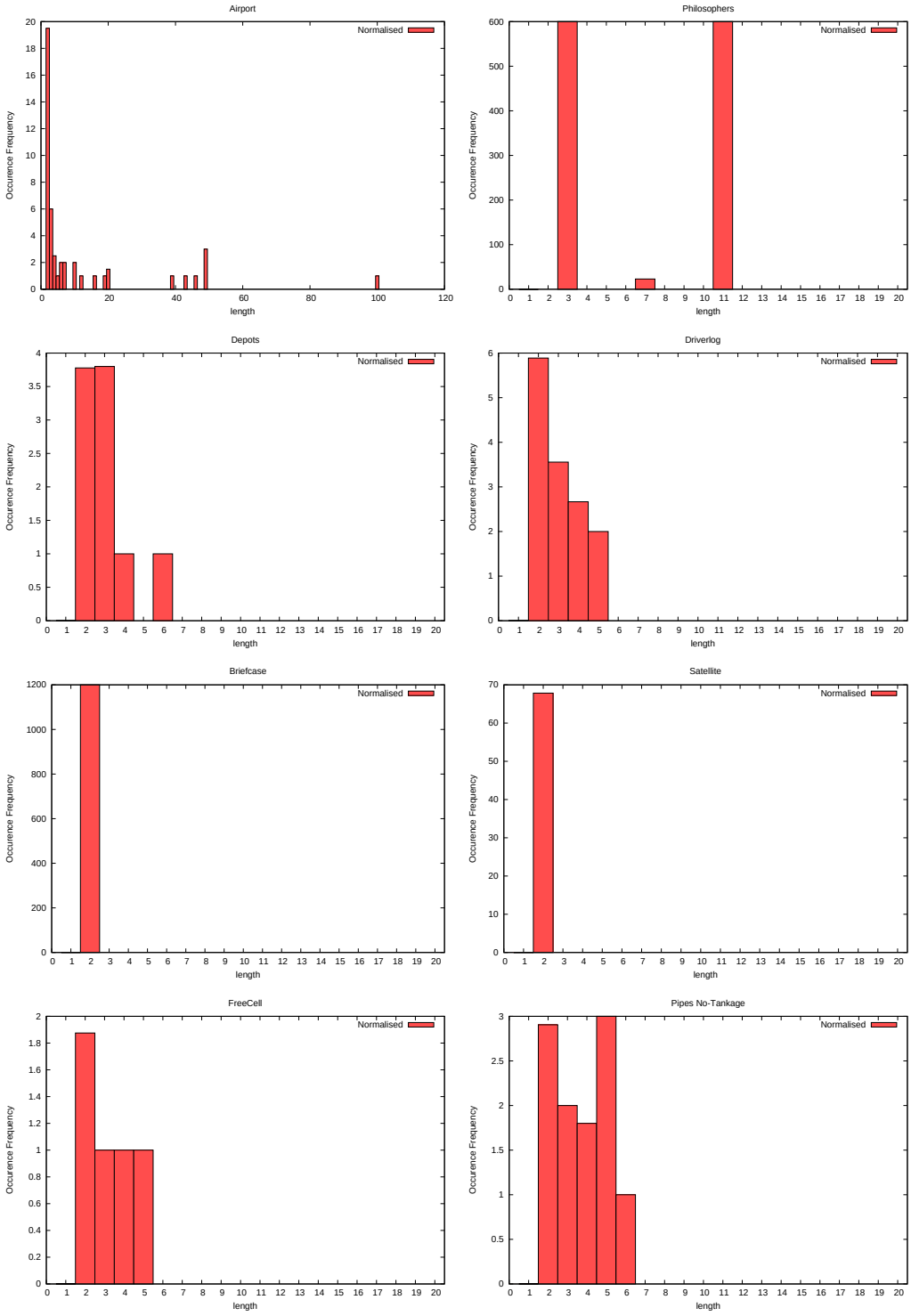


Figure 5.6: Mean frequency of use of macro-actions of varying lengths

length of macro-actions. A flexible system that can dynamically select which macro-actions to use can, however, allow a thorough investigation of the characteristics of useful macro-actions without disregarding certain classes of macro-actions. Existing systems which use a length cut off for potential macro-actions often insist on a maximum length of two for macro-actions, examples are the CA-ED version of Macro-FF [8] and the REFLECT system[15].

Hypothesis

Macro-actions of varying lengths can lead to improved search performance in terms of time taken to solve problems and number of problems solved. Macro-actions of length 2 are not sufficient to capture all the information that is useful in solving problems in many domains.

Analysis

Figure 5.7 shows the number of macro-actions of each length generated in each domain in the final library generated after keeping all macro-actions across all problems. Figure 5.6 shows the mean number of uses of macro-actions of each length. It can be seen from the results that plateau-escaping macro-actions of length 2 are used the most frequently but it is interesting to note that macro-actions of other lengths are quite often used. Furthermore logic suggests that use of a longer plateau-escaping sequence will result in a greater performance improvement as more exhaustive search is potentially avoided.

In the Airport domain it is most common to meet plateaux of length two; this results in the generation of 19 macro-actions of length 2. These macro-actions of length 2 are also the most used in this domain with each, on average, being used 10 times. In fact the most used macro-action is one of length 2 which is used a total of 56 times. The short plateaux cause short macro-actions to be generated and when these plateaux reoccur short macro-actions are used to escape them. It is, however, worth noting that many macro-actions of different length are also used: indeed a macro-action of length 100 is used once.

The observations made in the Philosophers domain are, however, rather different. The two equally most used macro-actions are of length three and eleven respectively. The approach of pruning macro-actions of length greater than two would clearly not allow the benefits of using macro-actions to be seen in this domain. This is a good example of a domain where the useful macro-actions are significantly longer than length 2, indeed no macro-actions of length 2 are generated. The highly symmetric structure present in this domain allows the planner

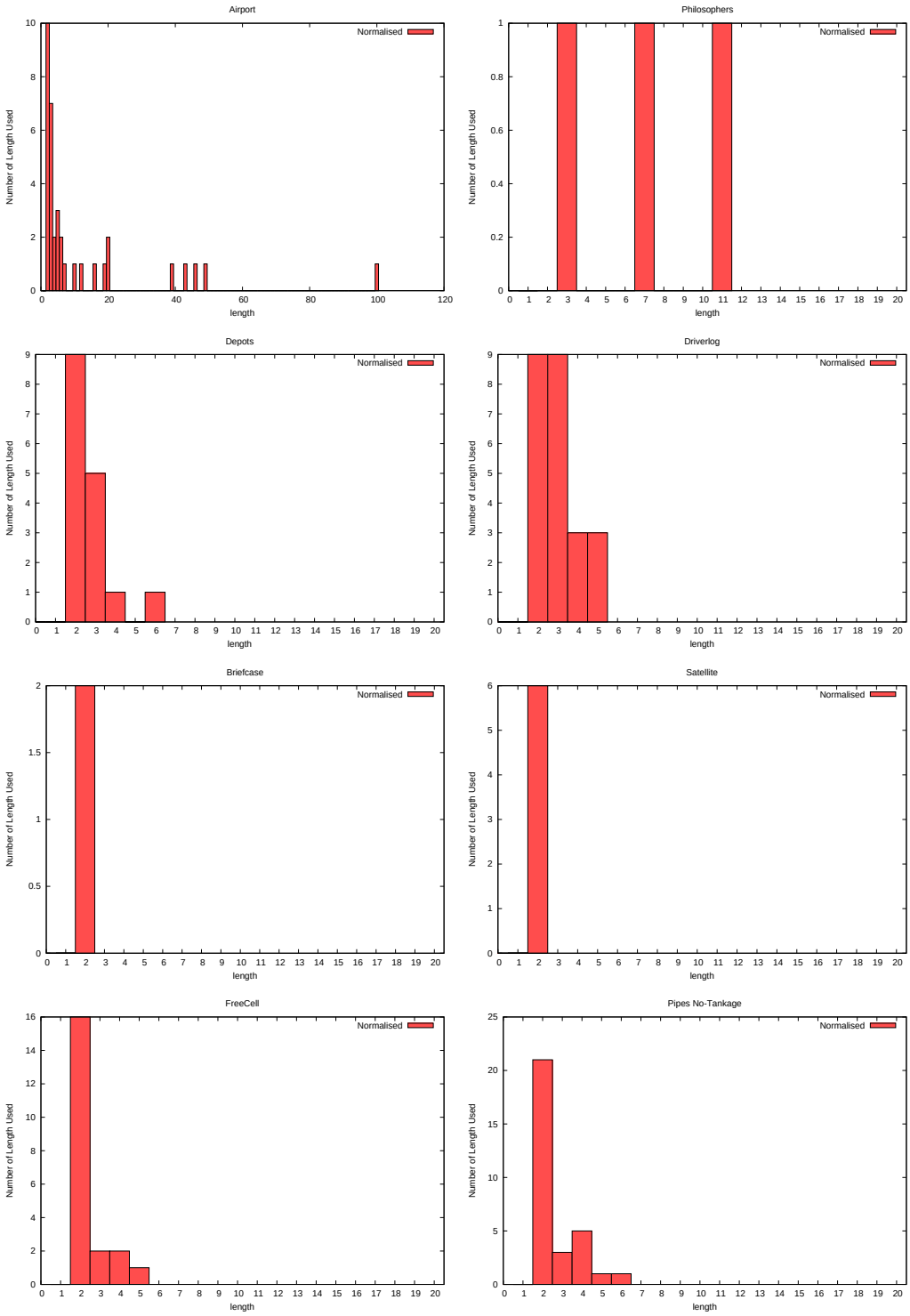


Figure 5.7: Number of occurrences of macro-actions of varying lengths

Length	Number of Times Used
2	3156
3	698
4	23
5	13
6	6
7	25
10	2
11	600
12	1

Table 5.10: Usage data for macro-actions of length up to 15. Missing data for a given length means no macro-actions of that length are generated.

to exploit longer macro-actions repeatedly in order to solve the planning problem much more quickly. The same 11-step macro-action broken into several 2 step macro-actions (clearly with one action missing out) would not be as effective in solving the problem: the planner would have to do search, with an increased branching factor, to order those two step macro-actions and escape a plateau.

In the Depots domain macro-actions of length three are the most commonly generated; it is, however, again the macro-actions of length 2 that are used the most on average. Macro-actions of length 3 are, however, still used on average 5 times each. Indeed, the second most used macro-action, used 10 times, is a macro-action of length 3; this is an extension of the most used macro-action lift-load (used 13 times) with the addition of the drive action to the end. The Driverlog domain shows that macro-actions of length 2 and 3 are the most abundant; with macro-actions of length 4 and 5 also being generated. Again the most popularly used macro-action length is 2 but macro-actions of length 3 are used, on average, more than 3 times each. Again the most used macro-action of length 3 is an extension of the overall most used macro-action that consists of 2 actions with the new action being added at the start of the macro-action: drive_truck, unload_truck, drive_truck.

The results in the Pipes No-Tankage domain show that macro-actions of length 5 are on average used the most. This is, in fact, just one macro-action of length 5 that is ranked seventh overall in the final library. Despite the fact that some macro-actions of shorter lengths are ranked above this macro-action; there are many macro-actions of shorter lengths that are not used very often, meaning that the average use for these macro-action lengths is smaller. In the FreeCell domain macro-actions of length to 3–5 are used once on average; the macro-actions of

	Max 5	**Max 4**	**Max 3**	**Max 2**	**No Caching**
Keep All					
(p =)	$1.464 * 10^{-08}$	$8.148 * 10^{-08}$	$8.348 * 10^{-09}$	$1.836 * 10^{-08}$	$5.897 * 10^{-05}$
Sig?	Yes	Yes	Yes	Yes	Yes
Best	Keep All	Keep All	Keep All	Keep All	Keep All
Max 5					
(p =)		0.01716	$2.81 * 10^{-05}$	$7.03 * 10^{-06}$	$4.77 * 10^{-08}$
Sig?		Yes	Yes	Yes	Yes
Best		Max 4	Max 5	Max 5	No Caching
Max 4					
(p =)			$3.627 * 10^{-07}$	$5.231 * 10^{-07}$	$6.866 * 10^{-08}$
Sig?			Yes	Yes	Yes
Best			Max 4	Max 4	No Caching
Max 3					
(p =)				0.06847	$1.846 * 10^{-11}$
Sig?				No	Yes
Best					No Caching
Max 2					
(p =)					$3.245 * 10^{-12}$
Sig?					Yes
Best					No Caching

Table 5.11: Significance table for the length pruning strategy: p is the probability that the null hypothesis cannot be rejected; sig? denotes whether or not the null hypothesis, that the versions take the same time to solve problems, can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

length 2 are used 1.8 times on average. These two domains are, however, domains in which macro-actions are not always useful in improving search performance.

In the Satellite and Briefcase domains the only macro-actions that are generated are macro-actions of length 2. All plateaux in these domains are of length two so no other macro-actions are generated. These actions are sufficient in themselves to escape plateaux in their respective domains and any actions that are required to follow them can be quickly found through heuristic guidance. The fact that only macro-actions of length 2 are generated and used in these domains is a result of the macro-action generation strategy and the structure of the domains themselves; rather than an indication of the fact that macro-actions of length 2 are best in the general case.

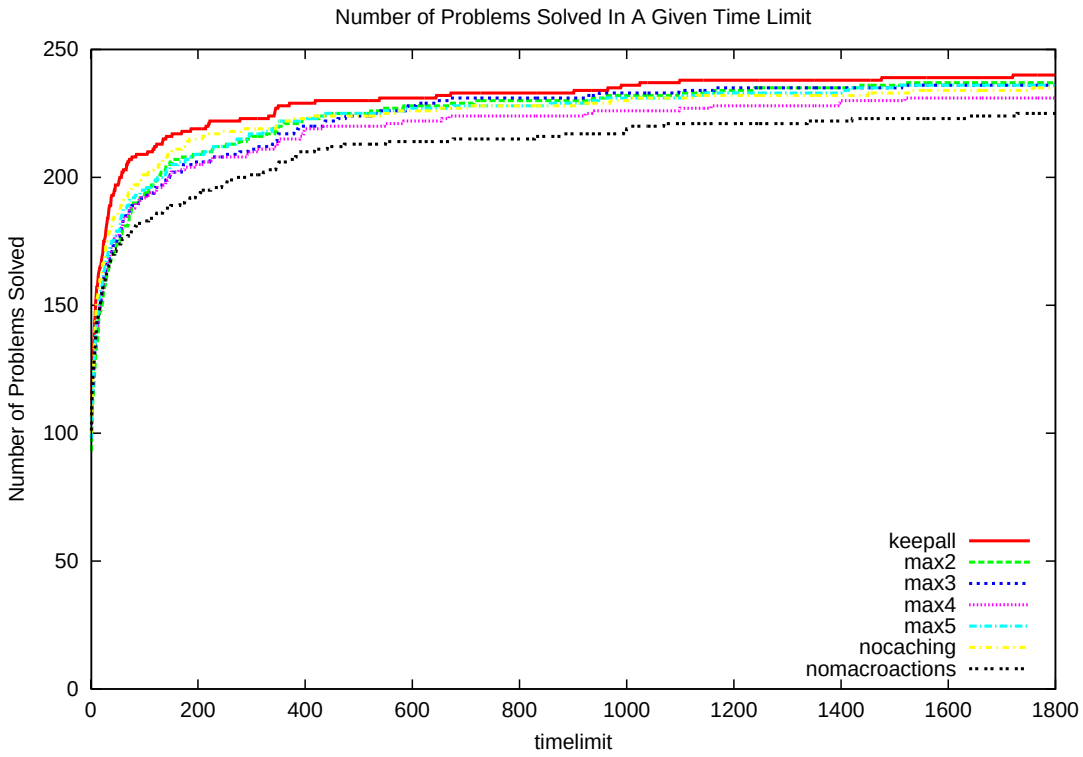


Figure 5.8: Percentage of Problems Solved in a Given Time Limit Keeping Only Macro-Actions Below a Given Length

Conclusion

Macro-actions of length 2 are the most popularly used macro-actions. It is, however, noteworthy that in this analysis that macro-actions of length greater than 2 account for 30% of macro-action usages. Figure 5.8 shows the result of only planning with macro-actions with a maximum length cut off. Each version of the planner is discarding macro-actions of length greater than the maximum specified. The planner is caching all macro-actions generated that do not exceed the maximum length cut off. It can be seen that keeping all the macro-actions is the most successful strategy, solving the most problems, as it allows the planner to consider all possible macro-action lengths. Further, figure 5.11 shows that the keep all version of the planner shows significantly better performance (in terms of time taken to solve mutually solved problems) than all of the versions of the planner pruning based on macro-action length. This, therefore, confirms the hypothesis that it is not wise to simply discard all macro-actions of length greater than 2 as those keeping those of other lengths as well can significantly improve planner performance. As shown in table 5.10 macro-actions of greater length can be, and frequently are, used in search leading to improved performance.

5.3 Strategies for Caching Macro-Actions

In this section the library management strategies described in section 4.2 are evaluated. Each of the strategies is evaluated with a number of different parameters. It is not possible to exhaustively investigate the whole of the possible parameter space, these are theoretically infinite (but in practice bounded by the maximum length reached by the library), and each possible parameter requires tests to be run on every problem across all of the domains. A fixed small number of parameters have been chosen for evaluation. The chosen parameters are more densely packed in the small end of the range, this is because as larger numbers of actions are kept in, or chosen from, the library the performance converges more and more towards keeping the whole library. Changes in the smaller ranges of the parameter space therefore have a greater performance impact. Note that the number of macro-actions that the planner must consider is much greater than the number of macro-actions generated: each macro-action can be instantiated many times with different parameters.

5.3.1 Search Time Pruning

Hypothesis

An effective search time pruning strategy can improve planner performance, minimising the potential for macro-actions to have a negative impact on search. The search time pruning employed is helpful macro-action pruning, described in section 3.5.1 and symmetric action pruning, described in section 3.5.2.

Analysis

The graphs in figure 5.9 show the results of running Marvin, caching all macro-actions, with and without search time pruning of macro-actions. Recall that the effect of applying macro-actions before or after unit actions was evaluated in section 5.2.2 and the results showed that applying macro-actions after all other actions was shown to be the best strategy. If macro-actions were applied before all other actions the search-time pruning would become even more critical.

The effects of search time pruning in the Airport domain are only slight, although the version using search time pruning does successfully solve two more problems. The limited effect is a result of the fact that many of the problems in the Airport domain are solved by best-first search, in which helpful action pruning is not used.

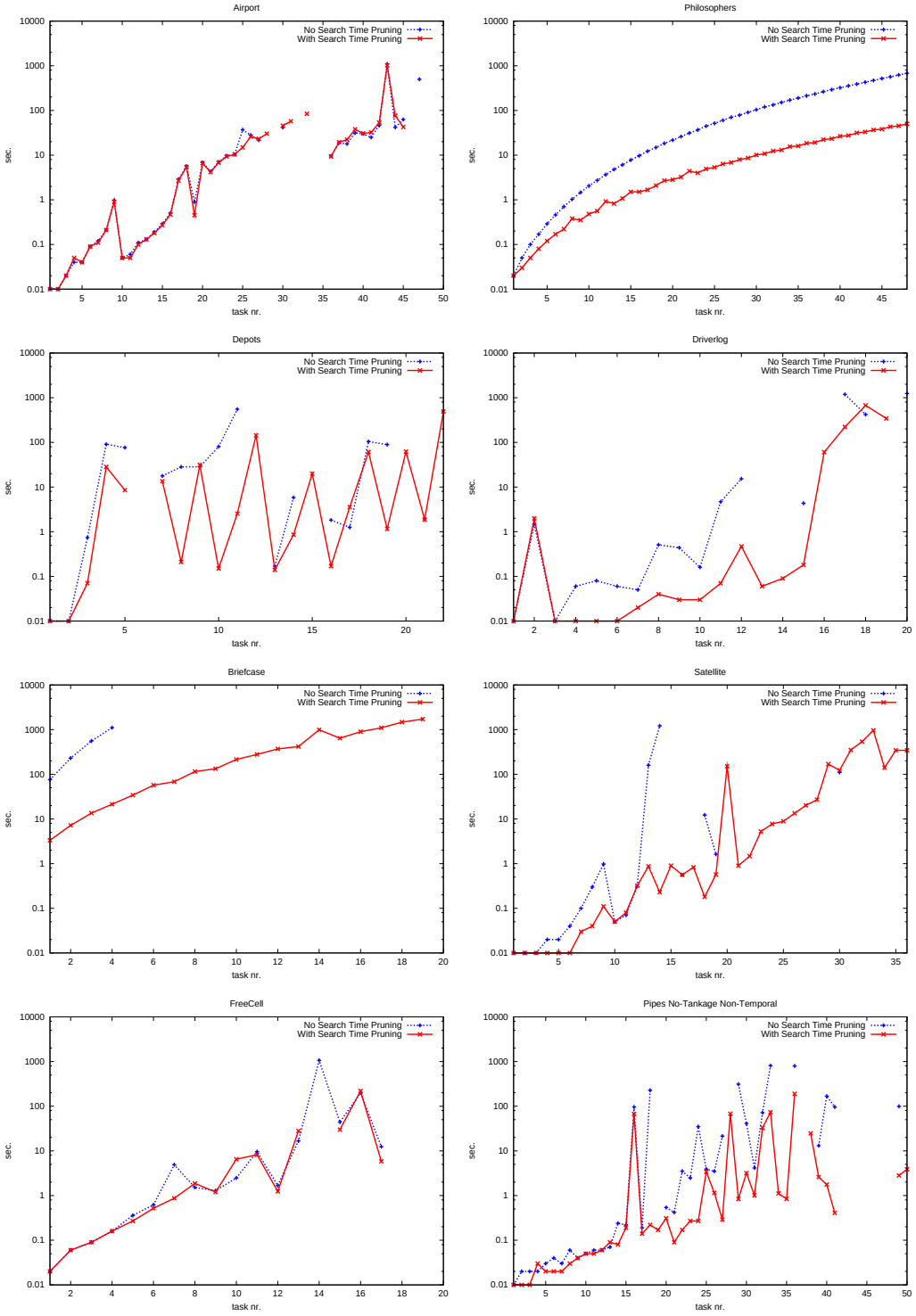


Figure 5.9: Time taken to solve problems in the evaluation domains using the keep all version of the planner with and without search-time pruning (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

In the Philosophers domain the effects of search time pruning are very noticeable. The version using search-time pruning solves all problems more quickly and is scaling much better. It is the helpful macro-action pruning that is allowing the benefits of using macro-actions to be seen. Without helpful macro-action pruning in this domain the planner must consider many useless actions before arriving at the correct action choice; with the pruning the planner explores far fewer actions before finding one that improves the heuristic value.

The Depots domain shows a rather different picture to the other domains with some problems being solved by the version using no pruning that are not solved by the version using search-time pruning. Both versions do, however, overall solve the same number of problems. Further, the version using search-time pruning solves the mutually solved problems, on average, 139 seconds faster. The reason that the versions solve different problems is that they resort to best-first search on different problems. In the absence of pruning the planner can select actions that otherwise would have been pruned allowing it to solve some problems without resorting to best-first search. It is, however, also the case, that in not pruning actions different actions may be selected, because they happen to be considered first, and these may not necessarily represent a good way to progress to the goal; this phenomenon causes the planner to resort to best-first search on two problems where it otherwise did not, and consequently fail to solve the problem. The makespan of plans generated in this domain is, on average, 25 steps longer in the no pruning version; this is also a result of the planner selecting different actions which, improve the heuristic value but are not necessarily on a path that leads as directly to the goal. In not using search-time pruning the planner loses the guidance taken from the relaxed plan in the form of removing macro-actions that do not lead to the goal efficiently.

In the Driverlog domain the planner solves problems more efficiently when performing search-time pruning, and solves many more problems. Only one problem, problem 20, is solved by the no pruning version that is not solved by the pruning version; in this case a macro-action that is normally pruned is needed in order to solve the problem within the time limit. A similar phenomenon occurs in problem 18 with the no pruning version able to use a normally pruned macro-action to solve the problem slightly more quickly. In all other problems, however, the pruning version is much more efficient as all of the macro-actions that are not pruned cause the planner to have to evaluate many more states before finding the correct one to select.

In the Briefcase domain all problems are solved by EHC and the planner is

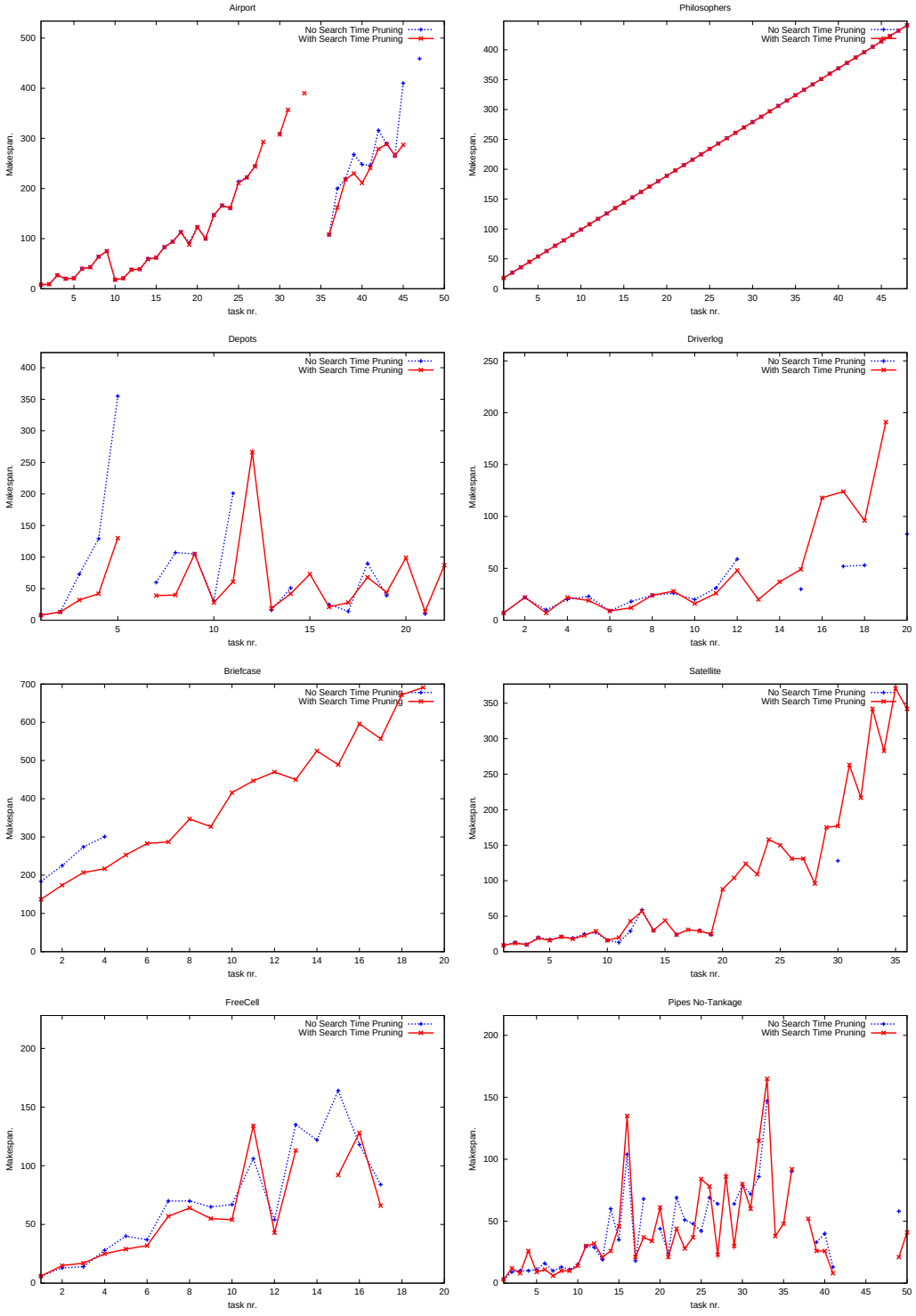


Figure 5.10: Makespan of Plans Generated Running the Keep All Version of the Planner With and Without Search-Time Pruning (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

able to solve many more problems when using search time pruning. Although only two macro-actions are generated in this domain, there are a very large number of possible macro-action groundings at every point and helpful macro-action pruning allows the planner to select only amongst the most useful. The performance in the Satellite domain is similar to this with the planner solving all problems when doing search time pruning but solving fewer than half of the problems. Again this domain has many possible groundings of macro-actions applicable in most states, not all of which usefully lead towards the goal.

FreeCell is the only domain in which the no pruning version solves one more problem than the pruning version. On this single problem the absence of helpful macro-action pruning allows the planner to solve the problem using EHC; the version pruning macro-actions prunes the necessary action very early in planning and is therefore forced to resort to best-first search. The performance on the other problems in this domain is similar for both versions, sometimes the pruning version finds a solution more quickly through avoiding considering macro-actions that are not useful; other times the no pruning version selects an action that would otherwise have been pruned and is able to find a solution more quickly.

The performance in the Pipes No-Tankage domain reflects that in the earlier domains. The version using search-time pruning is able to solve problems more quickly, through avoiding considering irrelevant actions. The makespan of the plans generated by the version using search-time pruning is, on average, 2 steps shorter.

Conclusion

Search time pruning allows the planner to solve many more problems. The helpful macro-action pruning allows the planner to take extra guidance from the relaxed plan generated in the heuristic computation. The macro-actions whose first step appear in the first level of the relaxed plan are considered for application: these are the macro-actions that the relaxed plan would suggest leads to a solution plan. When pruning all actions except these the planner is able to find solutions overall more efficiently by avoiding considering many macro-actions that are not useful in the current state. The pruning does occasionally lead to a few problems not being solved; but the benefits obtained by far outweigh this effect. The version using search-time pruning solves 240 problems; whereas the version not using pruning solves only 196. Performing a Wilcoxon signed-rank significance test on the data, with confidence level 95%, shows that the version using search time pruning solves problems in significantly shorter time than the no pruning version

(probability value $8.704 * 10^{-16}$) confirming that the hypothesis posed is correct.

The makespan of plans generated by the pruning version, shown in figure 5.10, is 12 steps shorter on average than that of plans generated by the no pruning version, as the planner sometimes selects macro-actions that would have been pruned and do not lead to a solution plan as directly. The helpful macro-action pruning allows the planner to take guidance from the relaxed plan to find shorter paths to the goal. Again the Wilcoxon signed-rank test shows the makespan of the plans generated by the pruning version to be significantly shorter to a confidence level of 95% (probability value 0.0007196).

5.3.2 Survival of the Fittest

As this is the first library management strategy to be considered much of the explanation of the effects of using macro-actions in each domain are described in this section, as well as the results of running the planner using the various configurations of the survival of the fittest strategy. The analysis of the remaining strategies is then done with reference to the comments on the effect of using macro-actions made in this section.

The motivation for the following hypothesis is that pruning macro-actions that have not been used reduces the overheads incurred by reasoning about a large number of macro-actions. Keeping only the most used macro-actions will allow the most useful macro-actions to be kept. This will not limit the performance of the planner by removing potentially useful macro-actions; yet will ensure that the performance of the planner does not degrade significantly by the introduction of a large collection of useless macro-actions. Keeping a very small number of macro-actions will improve the performance of the planner; keeping larger numbers of macro-actions will improve the performance of the planner further until a certain threshold is reached, beyond which the macro-action library size will become too big, with useless macro-actions being kept causing planner performance to degrade.

Hypothesis

Pruning Macro-Actions from the library based on use count will improve planner performance: limiting the number of actions to be considered, whilst still keeping the most useful.

Analysis

Figure 5.11 shows the time taken to solve problems in the evaluation domains keeping different numbers of actions in the survival of the fittest strategy. The performance is variable across the evaluation domains with different configurations exhibiting the best performance on different domains. Which configuration of the planner performs well—whether it be one of the caching strategies, the version using no macro-action caching or the version using no macro-actions at all—depends on the structure of the domain. Overall the version of the planner using the survival of the fittest strategy with the top 10 macro-actions solves the most problems; whilst the version using the same strategy caching the top 5 macro-actions solves the problems on average the fastest based on problems solved by both the strategy to be evaluated and the control version of the planner, as shown in table 5.13.

Airport

Upon first glance the performance of each of the strategies in the Airport domain appears to be more similar than in other domains. Many of the problems in this domain are, in fact, eventually solved by best-first search: the planner reaches a deadlock situation in which two planes are mutually blocking each others' forward progress (planes cannot reverse). Of the 38 problems solved by the control version 19 are solved by best-first search. The time taken to solve problems is therefore quite similar as the plateau-escaping macro-actions are most effective during EHC allowing the planner to skip over similar plateaux rather than in best-first search where, although they are used, the planner does not make a direct commitment to following the path of a good macro-action and will consider other alternatives if they appear heuristically good.

Macro-Actions can, however, give some improved coverage: in the version keeping all macro-actions 41 problems are solved and only 15 of these are solved using best-first search. The good search guidance offered by the macro-actions is sufficient to allow the planner to solve more problems using EHC avoiding the deadlock situation reached by the planner in the control version. The macro-action 'move-move' having only one aeroplane as a parameter is used frequently in these problems, this allows the planner to focus more on achieving the goals for one plane and moving it to the runway; rather than trying to move all the planes at once. In moving focussing more effort on moving one plane at once the deadlock situation is less likely to arise, explaining the success achieved by using this macro-action. The strategy of focussing effort on one goal has been

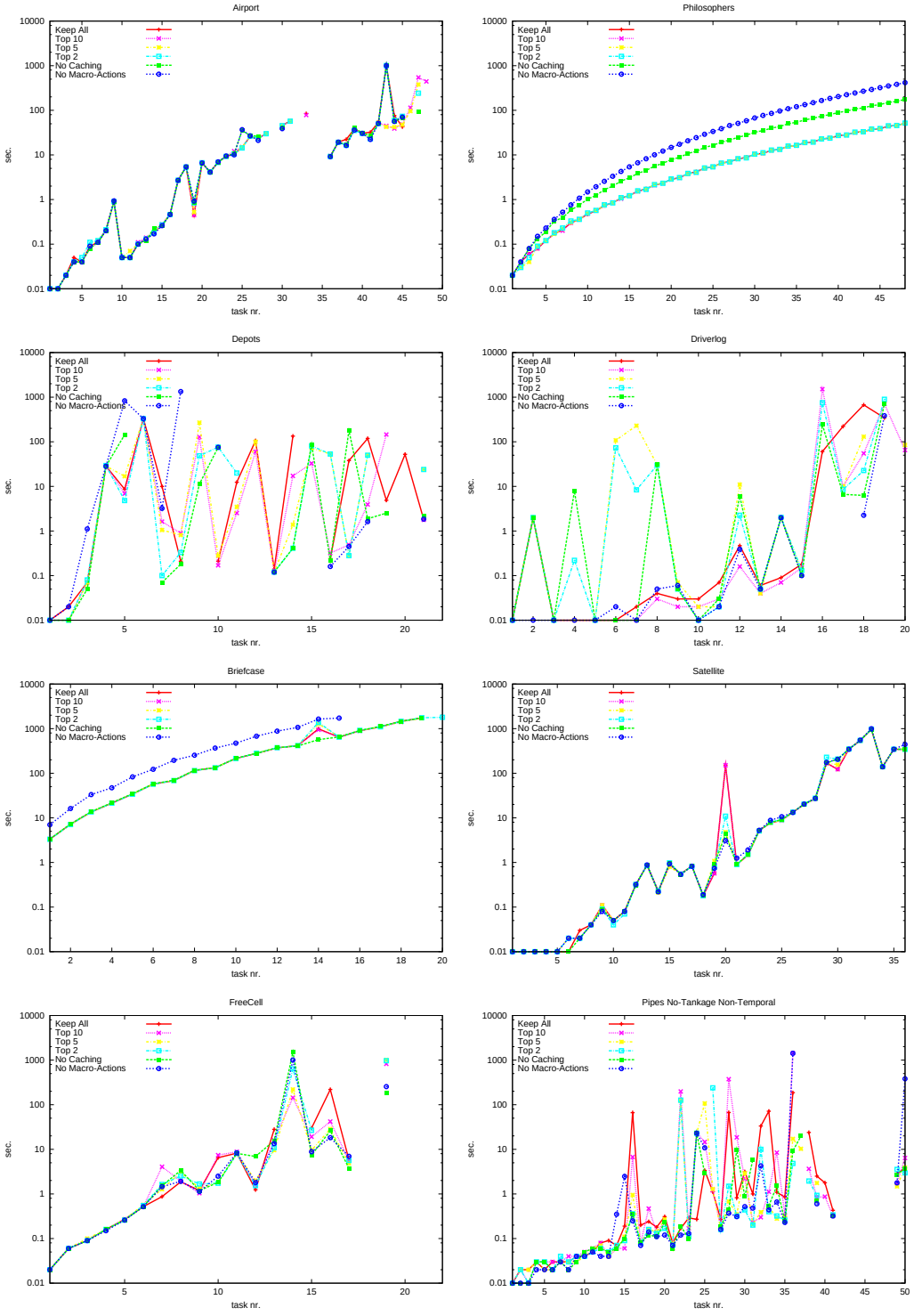


Figure 5.11: Time taken to solve problems in the evaluation domains adjusting the number of macro-actions kept in the survival of the fittest strategy (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

shown to be successful in this domain by SGPlan [31]; here the planner has learnt this. Sometimes it is necessary to move two planes to escape a plateau quickly, as a result of the state reached due to the way that search has progressed so far. The planner does learn a macro-action to do this but it is not as useful as the macro-action to move one plane twice so it is only considered if moving a single plane twice does not appear to be useful. The macro-action 'pushback-startup' is another popular macro-action in this domain. It is frequently used and allows the planner to plan two adjacent steps which clearly represent a logical sequence of events that must occur in order for an aeroplane to be ready to take off.

Comparing the relative performance of each of the caching strategies it can be seen that keeping all macro-actions causes problems, solved by this configuration and the control, to take ever so slightly longer. Keeping only the top 10 macro-actions allows the planner to avoid considering as many useless macro-actions that are not necessary for solving the problem. The macro-actions that can be generated in this way and are useful in solving these problems are contained within the ten most used macro-actions so the planner benefits from not having the need to consider other, less useful, macro-actions. When the planner only maintains the top 5 macro-actions the performance on instances solved by that version and the control is improved slightly, the planner is still benefiting from using the macro-actions to speed up search. The top 5 version does, however, not manage to solve as many problems as the top 10 version, it is forced to resort to exhaustive best-first search on two additional problems and therefore does not manage to solve the problem within the time limit. The top 2 version of the planner does not manage a great improvement in performance over the control version on mutually solved problems; it does, however, successfully solve three more problems within the 30 minute time limit. The macro-actions kept by the top 2 strategy (the 'pushback-startup' and 'move-move' with one aeroplane actions discussed above) are sufficient to aid the planner in avoiding deadlock situations and solving additional problems using EHC; but do not offer significant performance improvement in this set of problems if the planner could otherwise avoid this. Increasing the time cut off for each run of the planner or looking at different problems may allow such an improvement to be seen. The version using no macro-action caching can successfully solve one more problem than the version using no macro-actions; there is not sufficient information about deadlock avoidance gained in one run of the planner to allow it to achieve the same performance as the caching strategies. Overall performance is degraded very slightly due to the overheads of considering macro-actions that are not sufficiently well ordered

Domain	Keep All	Top 10	Top 5	Top 2	No Caching	No Macro-Actions
FreeCell	16	18	18	17	18	18
Airport	41	44	42	41	39	38
Depots	19	20	19	18	17	14
Philosophers	48	48	48	48	48	48
Driverlog	19	20	18	19	19	17
Pipes NT	42	42	41	40	39	39
Briefcase	19	19	19	20	19	15
Satellite	36	36	36	36	36	36
Totals	**240**	**247**	**241**	**239**	**235**	**225**

Table 5.12: Coverage Across Evaluation Domains Using the Survival of the Fittest Strategy

and filtered to be useful.

Looking at figure 5.13 it is worth noting that there is very little variation in plan length in this domain when using any of the caching strategies. Table 5.14 shows that the makespan of the plans is slightly improved overall by the versions of the planner performing macro-action caching. The macro-actions are allowing the planner to take a route off a plateau which is shorter to that it would otherwise have found[4]. The version using no macro-action caching generates plans with identical makespans to those generated by the no macro-actions version (in problems solved by both configurations).

Philosophers

In the Philosophers domain plateau-escaping macro-actions are very successful. They allow the planner to skip over several large plateaux with the application of a single macro-action. The structure of this domain is very amenable to the use of plateau-escaping macro-actions with several similar plateaux being present, this can be seen in figure 5.2 and was discussed in section 5.1.2. A plateau arises in the search space corresponding to each pair of adjacent philosophers: as the number of philosophers grows so does the number of identical plateaux that the planner encounters in solving the problem. Furthermore the size of these plateaux increases with the number of philosophers as there are more symmetric action choices to consider. Marvin does not suffer from this problem due to the symmetric action pruning strategy it employs (see section 3.5.2), however, standard FF would suf-

[4]Note that Marvin performs best-first (least-bad-first) rather than breadth-first search on plateaux meaning that escape trajectories are not guaranteed to be optimal in length.

fer as a result of this. Excluding problems 1 and 2, which have slightly different structure, there are only three distinct plateaux encountered in all problems in this domain. One macro-action is used to overcome large plateaux for pairwise adjacent philosophers in all problems with an additional macro-action used once in every even numbered problem to account for the remaining odd philosopher (the problems begin with two philosophers so even numbered problems have odd numbers of philosophers). The remaining, shorter, macro-action is again used in all problems, once for each pair of philosophers.

It can be seen from the graph for this domain in figure 5.11 that the difficulty of these problems follows a regular trend and the time taken to solve problems is consistently better for the versions using macro-actions. The version using no macro-actions solves problems the most slowly. The version generating macro-actions on each problem instance, but not caching them for later use, consistently outperforms the no macro-actions version, table 5.13 shows a 45 second mean time improvement for this version. The no caching version of the planner achieves this performance improvement because it must only do search to escape the first plateau of each type (at most three plateaux) and then can use a macro-action to avoid the expensive exhaustive search required to escape subsequent plateaux. Exhaustive search to escape even one plateau is, however, expensive; the versions caching macro-actions are not even required to do exhaustive search to escape one plateau and can successfully solve the whole planning problem finding a strictly better state at each point in the plan.

When caching macro-actions in this domain the danger of generating a large library of macro-actions is not present: the total number of different plateaux encountered in this domain is 4 (1 is only ever encountered in the first 2 problems); the bound on the library size can be clearly seen in figure 5.14. The result of the small number of distinct plateau types in this domain is that no degradation in performance is seen when keeping all macro-actions since the maximum library size is small. The version keeping the top 5 and top 10 macro-actions are therefore performing in the same way as the version keeping all macro-actions: both versions will have an identical collection of macro-actions since neither will ever prune the macro-action library. The top 2 version will remove some macro-actions from the library, figure 5.14 shows that the macro-action library size grows above 5 after solving problem 2. In this version only the two most used macro-actions can be kept; the third is, however, less expensive to compute than the others and is only required once one even-numbered problems so little performance degradation is observed. The makespan of plans generated in this domain are identical

for all versions. The macro-actions are allowing the planner to find the same plan more efficiently.

The other of the two Promela model checking domains, the Optical Telegraph domain, was not selected as an evaluation domain for all strategies due to its similarity to the Philosophers domain. Figure 5.12 has been included here to confirm the similar behaviour in this domain when using and caching plateau-escaping macro-actions. The apparently abrupt break in the trend, when the planner begins to fail to solve problems before one would otherwise expect, is explained by the fact that the planner does not fail to solve the problem due to the time limit being reached, but instead the imposed memory limit is reached. Similar behaviour to that in the Philosophers domain is clearly visible in the macro-action caching versions of the planner with the caching versions of the planner showing a considerable performance improvement over the version using no macro-actions. The top2 version does not show as much improvement as the other versions because it learns only two of the three macro-actions that are frequently used in this domain. The top three ranked macro-actions have use counts that differ by only one: after any problem if the top macro-action has been used n times the second macro-action has been used $n-1$ times and the third $n-2$ times. In problem p each of the top three macro-actions is used $p+1$ times, except in problems 1 and 2 where the macro-actions have their relative rankings established.

Unfortunately for the version caching two macro-actions, the macro-action that it discards as the least used is also the longest of these macro-actions. This macro-action has 7 unit steps, the other two macro-actions contain 3 and 4 steps respectively. It is possible to see, therefore, that these macro-actions work best as a unit of three: indeed plans consist of a macro-action repeated several times, followed constant repetition of the form unit-action1, unit-action2, macro-action1, macro-action2. The version caching only two macro-actions must compute the plateau-escaping sequence of the macro-action of length 7 on every problem and thus does not perform as well as the other versions. The poor performance of the no caching version of the planner appears surprising initially. When learning macro-actions on a per-problem basis the planner learns a macro-action which, when used, causes EHC to fail and the planner to resort to best-first search. In the caching versions, however, additional macro-actions are extracted from the solution plan which are used instead of the problematic macro-action and the planner can solve all subsequent problems using EHC. The no macro-actions version performs better than the no caching version as it does not learn the

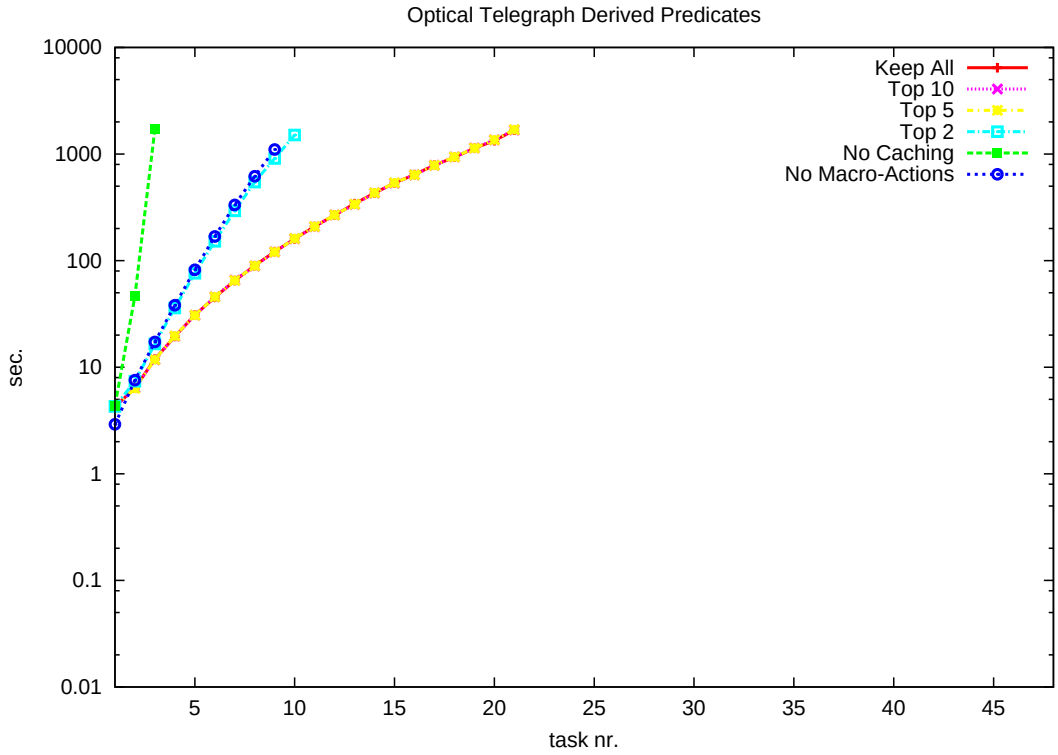


Figure 5.12: Using the Survival of the Fittest Caching Strategy in the Optical Telegraph Domain

problematic macro-action and can therefore solve problems using EHC. The plans produced by all versions of the planner are sequentially optimal and identical, as in the Philosophers domain.

Depots

Plateau-escaping macro-actions prove very helpful in the Depots domain. Unlike the Philosophers domain it does not have the predictable regular structure that shows that plateau-escaping sequences are highly likely to be reusable. It is, however, the case that similar plateaux do occur and macro-actions can be used to escape these efficiently. The reason for the dramatically improved coverage achieved by the versions of the planner using macro-actions is two fold. The first case is that macro-actions can guide the planner effectively to solve problems using EHC in many of the problems solved by the control using EHC. The second is that in many problems where EHC is destined to failure the macro-actions can lead the planner towards a recognised dead end more quickly so that the planner can then resort to best-first search and solve the problem, rather than continuing EHC search only to fail to escape a large plateau in the time limit. A significant proportion of the IPC problems in the Depots domain are easily solvable using

Domain	Keep All	Top 10	Top 5	Top 2	No Caching
FreeCell	-14.91 *(16)*	13.78 *(18)*	1.25 *(18)*	-21.38 *(17)*	-25.90 *(18)*
Airport	-0.61 *(38)*	26.31 *(38)*	26.42 *(38)*	0.04 *(38)*	-0.47 *(38)*
Depots	147.82 *(14)*	159.49 *(14)*	149.86 *(14)*	145.72 *(14)*	142.12 *(13)*
Philosophers	77.43 *(48)*	77.45 *(48)*	77.46 *(48)*	77.36 *(48)*	50.00 *(48)*
Driverlog	-36.98 *(17)*	-21.97 *(17)*	-31.72 *(16)*	-38.26 *(17)*	-21.89 *(17)*
Pipes NT	35.97 *(39)*	-6.14 *(39)*	40.12 *(39)*	43.73 *(38)*	46.89 *(38)*
Briefcase	281.14 *(15)*	284.43 *(15)*	261.10 *(15)*	254.91 *(15)*	308.74 *(15)*
Satellite	2.23 *(36)*	2.34 *(36)*	5.33 *(36)*	1.7 *(36)*	3.59 *(36)*
Totals	**61.51** ***(223)***	**66.96** ***(225)***	**66.23** ***(224)***	**57.98** ***(223)***	**62.88** ***(223)***

Table 5.13: Mean of time taken by no macro-actions version minus time taken using each configuration of the survival of the fittest pruning strategy. Results are calculated on mutually solved problems

best-first search but are difficult to solve using EHC. In the no caching version of the planner coverage increases by 3 to 17 problems solved and only two of these are solved by best-first search. Of the 18 problems solved by the top 2 macro-actions strategy 8 are solved by best first search; while the no macro-actions version solves only 14 problems, and 7 of these are solved by best-first search. What is noteable, however is that several of the new problems solved by the top 2 version and not by the no macro-actions version are solved because the planner resorts to best-first search more quickly. Furthermore, the top 2 version manages to successfully solve 4 problems using EHC that the no macro-actions version had to resort to best-first search in order to solve.

As the number of macro-actions cached in this domain increases the total number of problems solved increases for the bounded library sizes, as shown in table 5.12. The number of these problems solved by best first search, however, decreases from 8 to 5 to 3. When all macro-actions are kept 19 of the problems can be solved, slightly fewer than by the top 10 configuration. The keep all version does, however, solve all but 2 of these problems using EHC. The results

in this domain clearly show a peak in performance for the top 10 version. This is where the planner has reached the optimum point in the trade-off between keeping sufficient macro-actions so that good actions are not discarded; and pruning the library to remove macro-actions that are not useful. The patterns observed in the coverage achieved by the planner are also observed in the time to solve mutually solveable problems, shown in table 5.13.

The versions considering smaller numbers of macro-actions in this domain are not as successful as the version keeping the top 10. The plateaux in this domain are more varied than those in the Philosophers domain it is, therefore, not as clear which macro-actions will be useful in future problems, nor is it clear in which problem a macro-action will again be useful. By keeping enough of the macro-actions and pruning at search time, rather than pruning the library, the planner can dynamically select amongst a large enough collection of the macro-actions, using helpful action pruning, to find a useful macro-action generated on an earlier problem. The success of the top 2 caching strategy shows that a small number of plateau-escaping sequences do occur frequently; however, the further success of the versions keeping more macro-actions shows that rarely occurring plateau-escaping sequences are also helpful in this domain: the plateaux encountered here are more diverse than those in the Philosophers domain. The most used macro-action in this domain, was used 13 times by the end of the evaluation test; in contrast to the Philosophers domain in which the most used macro-action was used 600 times. The more varying structure in the Depots problems was observed in the heuristic profile discussion in section 5.1.

The solution quality of the plans generated by the macro-action caching strategies is superior. As discussed in section 5.2.1 when the planner uses macro-actions more problems are solved by EHC, where it is possible to reason about concurrency.

Driverlog

The performance in the Driverlog domain does exhibit some of the typical problems that are associated with the use of macro-actions in planning. It is possible to see from figure 5.11 that the performance on the easier problems in the domain is being made worse by the use of macro-actions in some versions of the planner. The strategies using macro-actions but caching only a small number, or none, of them are the worst affected. The spikes that can be seen in the graph occur in the problems where the macro-action versions are forced to resort to best-first search. The versions using macro-actions spend more time doing EHC before

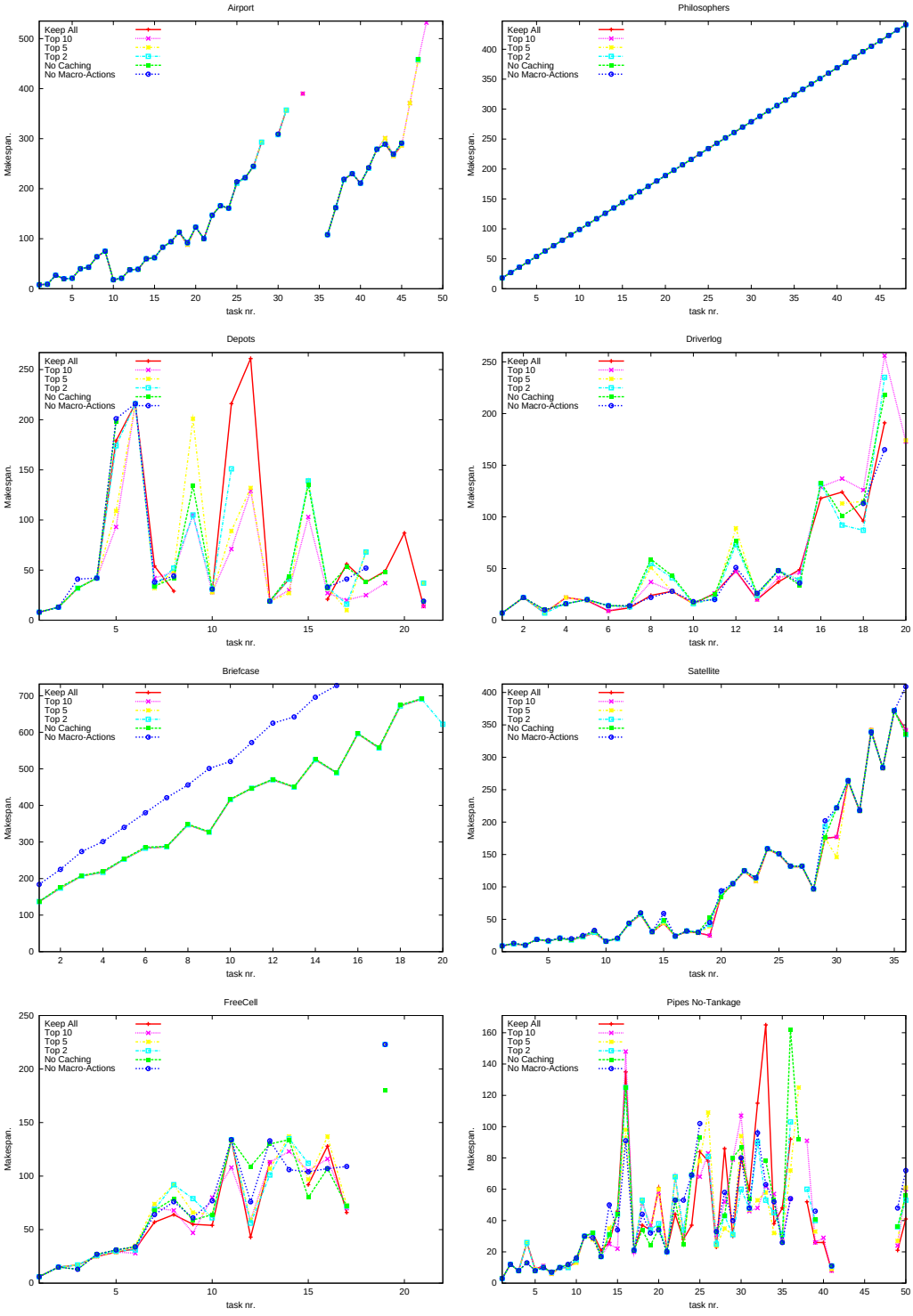


Figure 5.13: Makespan of plans Generated in the evaluation domains adjusting the number of macro-actions kept in the survival of the fittest strategy (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

reaching a dead end and resorting to best-first search. The top 2 and top 5 versions resort to best-first search on five problems: both on 2, 6, 7 and 14 and on 4 and 9 respectively. This accounts for most of the times the performance of these versions is worse than that of the no macro-actions version. The keep all and top 10 versions of the planner are not affected by this problem after problem 2; they have obtained a library of macro-actions that is sufficient to allow all problems that are solved by the control, and further problems not solved by the control, to be solved via EHC. The top 10 version of the planner solves all problems in the domain only resorting to best-first search on problem 2; the keep all version solves all but one of the problems, also only resorting to best-first search on problem 2; the control version, however, solves only 9 of the problems successfully using EHC.

The mean solution time for the problems, shown in table 5.13, appears to show that the use of macro-actions in this domain makes planning less efficient. Viewing this in conjunction with the graph in figure 5.11, in the larger problems in general the use of macro-actions does not degrade performance significantly. In fact, the performance of all the versions is similar (on problems other than those discussed in the previous paragraph) except that each configuration solves one problem, or in some cases two problems, considerably more slowly than the control (some problems are very easily solved by best-first search but not so by EHC). This makes the mean time difference large when in general the differences are quite small; whether the no macro-actions version or the other respective version is slower varies on a per-problem basis. Of course, if the version using no macro-actions was allowed to complete search, exceeding the time limit, on the problems it has failed to solve the mean time difference between it and each of the configurations would clearly show the other configurations to be faster. Overall the results show that using macro-actions in this domain is beneficial on problems that cannot be very efficiently solved by best-first search: one way to prevent the versions using macro-actions from exhibiting worse performance due to this behaviour is simply to allow the planner to do 10 seconds of best-first search before then proceeding with search in the normal manner. This would mean that some harder problems were solved 10 seconds more slowly (although this is less noticeable when the problem takes in excess of 1000 seconds to solve anyway); however, on the smaller problems the undesirable increases in time taken would not be seen.

Solution quality is made slightly worse when using macro-actions: the macro-actions that allow the planner to solve the problem using EHC as opposed to

best-first search do so at slight cost. The macro-actions do, however, allow the planner to solve more problems. Furthermore most of the quality degradation of the plans is in a few problems: although overall the sum of the plan lengths for the no macro-actions configuration is smaller than that for any other configuration; the no macro-actions version of the planner only solves 6 problems with shorter plans than the other versions (only 5 in the case of the keep all version). When the versions using macro-actions generate plans longer or shorter than the solution for the no macro-actions version it is often the case that this is a result of the search path taken for EHC, in the version using macro-actions, being different to that taken during best-first search, in the no macro-actions version. This means that the plans are quite different in structure, using different driver and truck combinations in different locations; rather than the macro-action plan being an inferior version of the no macro-actions plan containing redundancy.

In this domain the macro-action library does grow large for the keep all version of the planner (as shown in figure 5.14); but the search time pruning allows the planner to plan with a large macro-action library without a noticeable degradation in performance.

Briefcase

The results obtained for the Briefcase problems show performance similar to that in the Philosophers domain: a regular trend in difficulty can be observed in figure 5.11 and the versions using macro-actions outperform the version using no macro-actions. The difference between the results generated in this domain and those in the Philosophers is that the performance of the no caching version is much more similar to the performance of the versions doing macro-action caching. This can be explained by the nature of the plateaux in the respective search spaces. In the Philosophers domain the plateaux are larger, requiring 3, 7 and 11 steps to escape; in Briefcase problems, however, the plateaux never require more than two steps to escape. On the first plateau in problem 10 of the Philosophers domain 576 nodes are expanded; whereas the first plateau in problem 10 in the Briefcase domain requires only 123 nodes to be expanded. The difference becomes larger the harder the problem instance becomes.

It is, however still possible to see an improvement in the versions using macro-actions compared to the version that is not using macro-actions. This is because although the plateaux are short and not very expensive to escape, the occur so frequently that the sum of the small savings in time made by using macro-actions becomes significant. In Briefcase problem 1 there are 34 plateaux in contrast

Domain	Keep All	Top 10	Top 5	Top 2	No Caching
FreeCell	8.56 *(16)*	5.22 *(18)*	0.33 *(18)*	1.76 *(17)*	3 *(18)*
Airport	0.5 *(38)*	0.18 *(38)*	0.24 *(38)*	0.18 *(38)*	0 *(38)*
Depots	3.5 *(14)*	12.21 *(14)*	7.29 *(14)*	1.64 *(14)*	1.62 *(13)*
Philosophers	0 *(48)*	0 *(48)*	0 *(48)*	0 *(48)*	0 *(48)*
Driverlog	-0.16 *(17)*	-6.63 *(17)*	-4.31 *(16)*	-6.35 *(17)*	-7.88 *(17)*
Pipes NT	-2.41 *(39)*	0.44 *(39)*	2.85 *(39)*	0.24 *(38)*	-2.71 *(38)*
Briefcase	122.4 *(15)*	122.4 *(15)*	122.4 *(15)*	122.4 *(15)*	120.93 *(15)*
Satellite	5.86 *(36)*	5.86 *(36)*	6.56 *(36)*	3.11 *(36)*	3.25 *(36)*
Totals	**17.28** ***(223)***	**17.46** ***(225)***	**16.92** ***(224)***	**15.37** ***(223)***	**14.78** ***(223)***

Table 5.14: Mean of makespan of the solution plan generated by the no macro-actions version minus makespan of plan generated by each configuration of the survival of the fittest strategy. Results are calculated on mutually solved problems.

to the 2 found in Philosophers problem 1. The maximum number of plateaux found in a Briefcase problem is 234 in contrast to the maximum number in a Philosophers problem which is 49. The most used macro-action 'move_briefcase put_in' is used 1654 times in total across all problems by the top 2 version of the planner; the second most used macro-action 'move_briefcase take_out' is used 654 times. All other caching versions perform as the top 2 version of the planner: it can be seen from figure 5.14 that no more than two macro-actions are ever generated.

The point at which the no macro-actions version of the planner fails to solve more problems represents an interesting feature of the search using macro-actions. Clearly the trend predicts that the macro-actions version should not fail to solve the next problem within the time limit; the prediction is correct, it is not the time limit, but the memory limit, that prevents the planner from solving further problems. Macro-actions are therefore not only allowing the planner to solve the problem in less time but also to decrease the memory usage. This is logical as the time saved by using macro-actions is closely related to the saving in the

number of nodes that the planner has to expand in order to solve the problem. In solving problem 15 the no macro-actions version of the planner expands 48555 nodes; while the top 2 version of the planner expands only 19828. When the no macro-actions version of the planner exceeds the memory limit on problem 16 it has expanded 45968 nodes; the top 2 version of the planner solves the problem having expanded only 25505 nodes.

The solution quality of the plans generated in this domain is superior for the versions of the planner using macro-actions. This is because the no macro-actions version of the planner occasionally inserts redundant move actions, moving the briefcase from one location to another via a third location. Since any place is reachable from any other place, chains of move actions are not necessary in this domain. The relaxed plan length may, however, decrease in moving to such a location because the planner may believe that it can unload an object which is not yet in the briefcase. By moving the briefcase the destination location for the object the planner no longer has to add a move action to pick up the briefcase and one to drop it off: a move action to collect the object is sufficient as the fact that the briefcase is in the object's destination location will not be deleted in the relaxed plan. The version of the planner using macro-actions generates two macro-actions, one of which is a put_in action followed by a move_briefcase action, the other is a move_briefcase action followed by a take_out action. These two actions encourage the planner to do something useful once the briefcase has been moved to a certain location rather than just going via another location unnecessarily.

Satellite

The performance in the Satellite domain is, as expected, very slightly better for the versions using macro-actions than the no macro-actions version. As discussed in section 5.1 very few plateaux are encountered in this domain and those that are encountered are very short. The plateaux are, however, similar to each other so it is possible to reuse macro-actions and solve problems faster. The time improvements seen are not, however, great because the plateaux are infrequent and are short, meaning exhaustive search to escape them is not greatly expensive. Table 5.13 shows that overall problems are solved slightly more quickly by the versions using macro-actions than the no macro-actions version.

Problem 20 is the only problem in which the no macro-actions version of the planner finds a solution clearly noticeably more efficiently than any of the other versions of the planner. This problem is solved more quickly by the no macro-

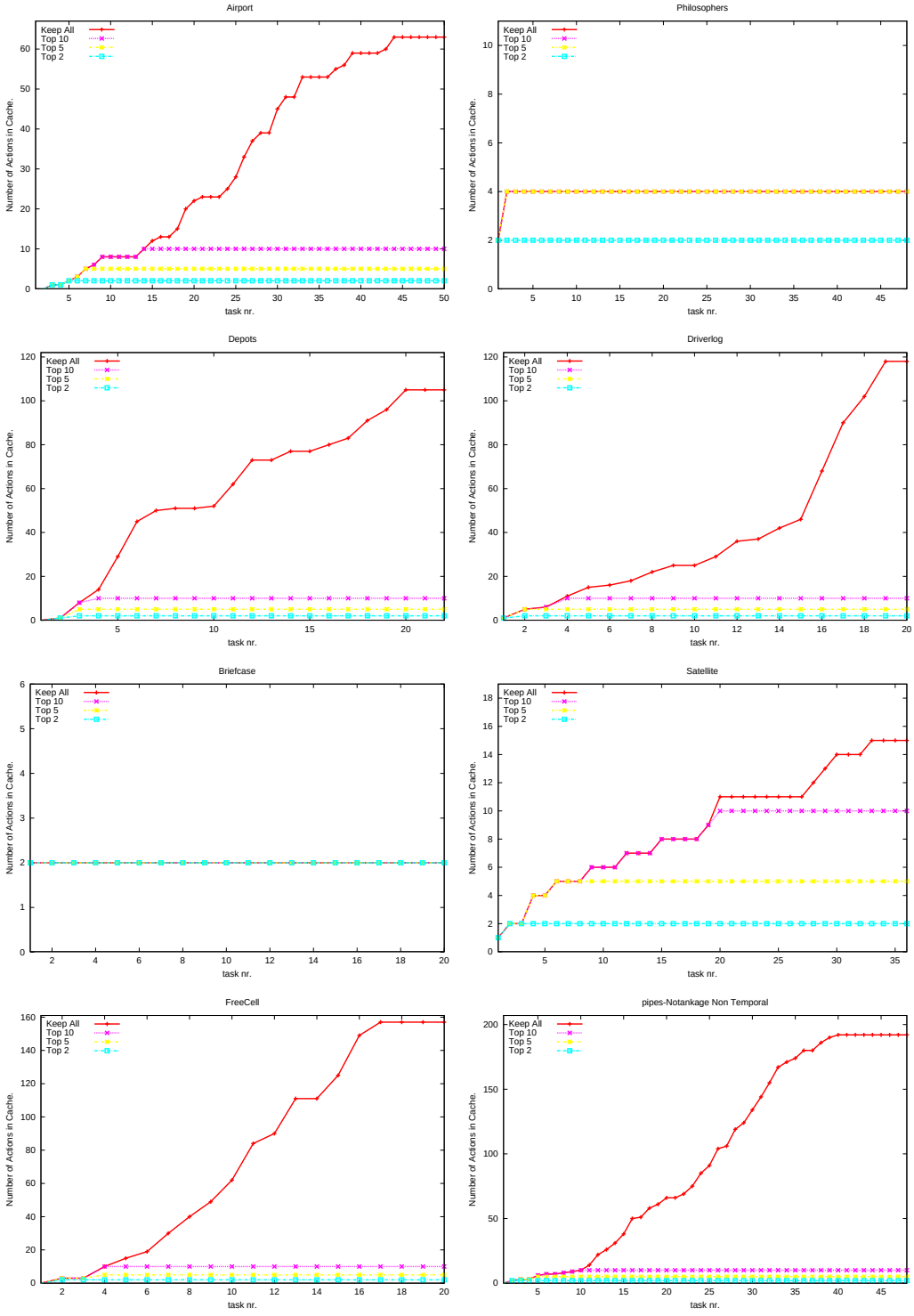


Figure 5.14: Number of macro-actions in the macro-action library when using different configurations of the survival of the fittest strategy. Results are shown on all evaluation domains (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

actions version due to the appearance of one state in which many macro-actions are helpful. The more macro-actions present in the library, the longer the planner takes to solve this problem. After the application of three actions (identical in all versions of the planner) a state is reached from which many of the macro-actions are helpful. This occurs in the Satellite domain because there is much symmetry in the problem and few mutexes between interdependent actions: many satellites can turn at once, or take an image at the time another one is turning etc. All of these actions will be in the first step of the relaxed plan. The graph in figure 5.11 shows a small spike for the no macro-actions version as it contends with the problem of having many helpful actions to choose between. This problem is amplified when the planner has a large collection of macro-actions, as these will also be helpful actions requiring consideration: figure 5.14 shows that by problem 21 the top 10 and keep all versions of the planner have 10 and 11 macro-actions in their libraries respectively. Some other problems are solved slightly more quickly by the no macro-actions version of the planner as a result of this phenomenon; however, in these cases the difference in time taken is much smaller as there are not quite so many helpful macro-actions present.

The phenomenon does not, however, occur so much that the macro-action versions of the planner are unable to solve problems faster than the no macro-actions version: indeed on average they solve problems more quickly despite the fact that the potential gains are limited by small plateau sizes. In a domain with similar structure but larger plateaux the planner would benefit more from the plateau-escaping macro-actions and this would further outweigh the costs of considering non-helpful actions. The plateaux in this domain are, fortunately, quite similar: that is, the keep all version of the planner has only generated 15 macro-actions by the end of solving the problem suite. This relatively small number of macro-actions (in contrast to in excess of 100 in many domains) further helps in avoiding the problem of having a large number of helpful actions.

The makespan of the plans generated when using macro-actions are again shorter than those generated without macro-actions. In problem 21, the problem in which the caching versions take significantly longer to solve the problem, the versions using macro-actions do generate shorter plans, albeit in a longer time. The improvement in makespan is only slight. It is, however, noteworthy that the use of macro-actions has again not only allowed faster planning without a decrease in solution quality; but has actually improved the quality of solutions to many problems.

FreeCell

The FreeCell domain was included as an evaluation domain due to the directed nature of the search space encountered when solving problems. The hypothesis was that macro-actions would not be helpful in this domain as they may lead search into unrecognised dead ends. Pleasingly the coverage in this domain has not decreased for several of the macro-action versions of the planner, as shown in table 5.12. The only versions which have failed to solve as many problems as the no macro-actions version are the keep all and top 2 versions. On the early problems the directed nature of the search space does not cause as many difficulties: the available space (free cells and empty columns) is less constrained so there are actually fewer dead ends. As the number of cards in the problem increases the available space becomes more constrained and dead ends appear more frequently.

The macro-actions alter the direction in which EHC proceeds quite dramatically, with each of the versions taking different paths through the search space on most problems. The performance of any given configuration of the planner on a given problem is determined by which macro-actions are learnt and which direction these macro-actions happen to lead search through the search space on the given problem. There seems to be no clearly identifiable 'good' macro-actions with the outcome being more dependent on chance decisions made rather than good, or bad, guidance given by the macro-actions. Indeed a macro-action that is useful on one problem can make planner performance worse on another problem. Sometimes the macro-actions offer good search guidance, allowing the planner to avoid resorting to exhaustive search; other times the macro-actions result in bad decisions being made and the planner reaching unrecognised dead ends. The time taken to solve mutually solveable problems reflects this unpredictability with some versions happening to make good decisions and solving problems, on average, more quickly; whilst other versions make bad decisions and solve problems more slowly. Tests performed on additional, full sized, FreeCell instances to attempt to gain a greater insight into the effects of macro-actions in this domain also reveal that little predictability can be seen.

Pleasingly very bad decisions are rare: there are only two problems that the keep all version does not solve that are solved by the no macro-actions version. On one of these problems it is simply the case that the no macro-actions version of the planner fails quickly to solve the problem using EHC, but the problem is sufficiently simple to be solved by best-first search. In this problem the keep all version of the planner has avoided the dead end and is still successfully performing

EHC when the time limit is reached. If the problem were more interesting, and not trivially solveable by best-first search this would work to the advantage of the keep all version. The other problem not solved by the keep all version is, however, a result of a bad decision made by the planner.

The makespan of the plans generated by the versions using macro-actions is shorter, on average, than that generated by the no macro-actions versions. As observed earlier the path through the search space taken when using macro-actions is sometimes better and sometimes worse, in terms of solution time, with no clear bias towards better or worse performance. In terms of makespan, however, using macro-actions does generally lead the planner to find a shorter path through the search space, allowing the generation of shorter plans.

Pipes No-Tankage

In this domain macro-actions are helpful in allowing the planner to avoid resorting to best-first search to solve the problem. The keep all version of the planner solves 42 problems; only one of these is solved by best-first search. This is in contrast to the no macro-actions version of the planner which solves only 39 problems of which 7 are solved by best-first search. In general when the no macro-actions version of the planner solves the problems more slowly than the keep all version it is a result of the no macro-actions version having to resort to best-first search. The macro-actions therefore make solving the problems via EHC generally slightly slower but more robust. In the problems that are not solved by the no macro-actions version of the planner, but are solved by the keep all version of the planner, the no macro-actions version of the planner has resorted to best-first search and then failed to complete within the time limit. In general it appears that EHC is a good planning strategy in this domain when the planner is able to make reasonably informed decisions; best-first search is, however, only successful on small problem instances. Indeed, a planner doing restarts based on a stochastic version of EHC has been shown to perform very well in this domain solving all problems [13]. Macro-actions can provide extra guidance and allow the planner to solve more problems by EHC thus increasing coverage.

The various configurations of the survival of the fittest pruning strategy perform better than the no macro-actions and no caching versions of the planner; but not as well as the keep all version of the planner. Keeping all of the macro-actions maximises the chance of finding a macro-action that allows the planner to avoid resorting to exhaustive search on a given problem. When fewer macro-actions are kept fewer of the problems are solved successfully by EHC, the planner resorts to

best-first search and fails to solve the problem.

Table 5.13 shows that all but one of the versions of the planner using macro-actions solve problems more quickly than the no macro-actions version. The time saving is mostly a result of the problems on which the planner does not have to resort to best-first search, and can instead solve very much more efficiently via EHC. Indeed the reason that the top 10 version takes a longer time to solve mutually solveable problems is due to the learning of a bad macro-action that causes the planner to resort to best-first search on two easy problems thus increasing the mean time. Some problems solved via EHC by the top 10 version and not the no macro-actions version of the planner do not count in the improvement as the no macro-actions version fails to solve them. If these problems were included an actual time improvement would be seen. On problems solved by EHC in both versions of the planner the no macro-actions version is quite often able to find a solution in a similar or shorter time.

The phenomenon of bad macro-actions is an interesting one, it appears that some macro-actions can cause search to complete more slowly. The longer times taken to complete search are, however, not a result of the increased branching factor, the problem often associated with potential negative effects of macro-actions; but instead are problems with macro-actions leading search into dead ends during EHC. When dead ends are reached the planner must resort to its secondary search algorithm: best-first search, which is generally not as efficient as EHC. The effect of macro-actions leading the planner into a dead end can be either positive or negative. If search using EHC is destined to fail the macro-action can lead the planner into the dead end more quickly, thus reducing the wasted search time and allowing best-first search to commence more quickly. In the case where EHC would otherwise have succeeded were the macro-action not included in the search the macro-actions are considered bad macro-actions: they have forced the planner to resort to best-first search, which is often more expensive, to solve the problem. Despite the occurrence of these bad macro-actions the use of macro-actions still provides an overall benefit. There are two reasons for this: the first is that bad macro-actions rarely occur so their impact on performance is not seen on many problems; the second is that the performance gains made by using macro-actions are often large and outweigh any negative impact of bad macro-actions.

The makespan of plans generated by each of the configurations is very similar in this domain with fewer than 3 time steps difference, on average between each of the configurations and the no macro-actions version. Very little concurrency

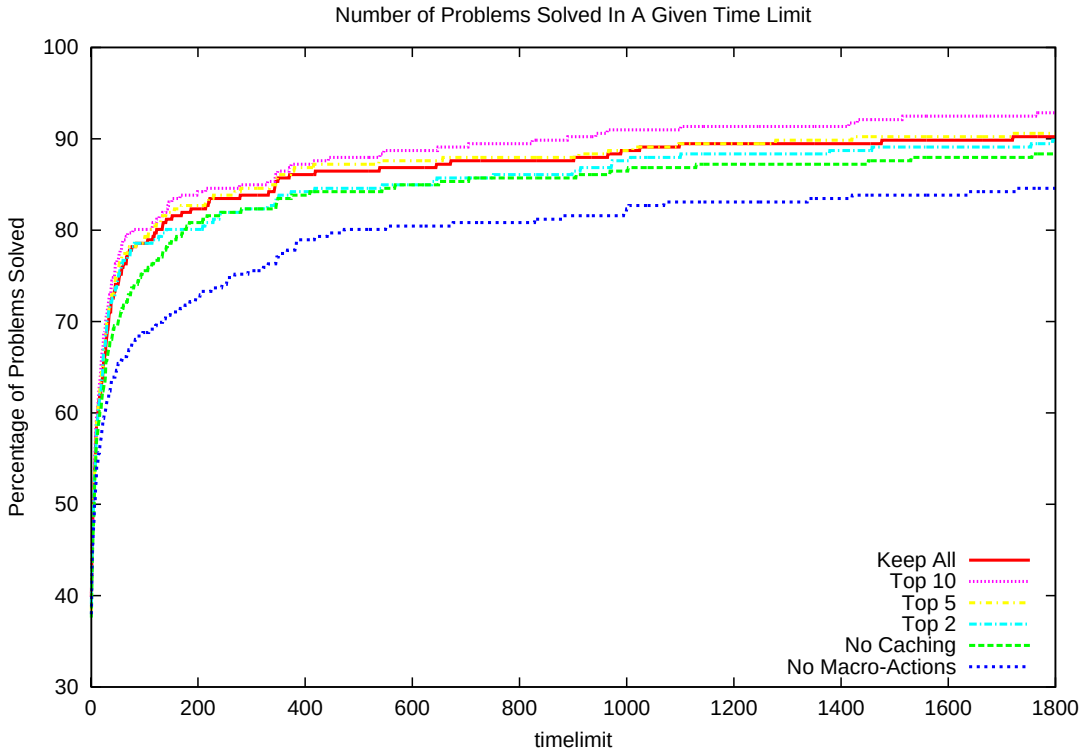


Figure 5.15: Percentage of Problems Solved in a Given Time Limit Using the Survival of the Fittest Caching Strategy

is observed in the plans generated by Marvin in this domain: the pipe networks are often highly interconnected, so performing actions often affects the state of entities affected by other actions, leading to temporal mutexes. The makespan improvements observed in other domains due to concurrent reasoning in EHC and not in best-first search cannot, therefore, be seen in this domain. The makespans are all similar: sometimes the different path taken through the search space using the macro-actions learnt by a given version yields a shorter plan; other times a longer plan. The solution quality is, however, on average not significantly different.

Conclusion

Pruning the macro-action library, keeping only the most used macro-actions, can improve coverage across the evaluation domains. A summary of the results is shown in figure 5.15, this indicates the percentage of the problems, across the whole evaluation suite, that the planner had solved within a given time limit. The version of the planner keeping the top 10 macro-actions solves the most problems overall; followed by the top 5 version which is, in turn, closely followed by the keep all version of the planner. As discovered in section 5.3.1 the search

time pruning employed is very powerful allowing the planner to keep a large number of macro-actions without great runtime costs. The pruning of the library can, however, allow the performance of the planner to be improved further. If the search time pruning is disabled a very different picture is observed using the survival of the fittest strategy: the no macro-actions version of the planner solves the most problems, 220, the performance then degrades with no caching solving 210. The coverage then decreases as the number of macro-actions kept increases the top 2 version is the best of the caching versions, solving 206 problems whilst the worse is keep all solving only 196 problems. It is clear to see, therefore, that the search time pruning is critical in determining which number of macro-actions is the best to keep over this evaluation suite.

Table 5.15 shows the results of performing a Wilcoxon signed-rank significance test on the data generated across all domains. The values in the table show the probability that the null hypothesis, that the versions take the same time to solve problems, is correct. The sig? rows show whether the null hypothesis can be rejected with 95% confidence. All macro-action-using versions of the planner have been shown to offer a significant improvement over using no macro-actions. Each macro-action caching version shows significant improvement over the no caching version of the planner. The results for the survival of the fittest strategy are pleasing with the top 10 and top 5 versions of the planner showing a significant improvement over the keep all version. This confirms the hypothesis that pruning the macro-action library using this strategy can offer significant performance improvements if the appropriate number of macro-actions are kept.

In Macro-FF [6] the top 2 macro-actions were kept (top two was defined differently according to their ranking system); here, however, we observe that it is better to keep more than two macro-actions. There are two reasons for this, the first is the search time pruning allowing macro-actions to be reasoned about only when necessary, enabling more to be kept. The second is that the ranking procedure in Macro-FF took place offline and could therefore reason about which is the best macro-action to keep in more detail.

To answer the question of whether the number of times a macro-action has been used is a good indicator of whether or not it will be useful in the future additional tests were performed. The problems in the evaluation suite were solved using a version of the planner that selects 10 random macro-actions to save in the library at the end of each problem run. This version of the planner is only able to solve 233 problems, not as many as the 247 solved by the top 10 version of the planner. This relative success of the top 10 version of the planner shows

	Top 10	**Top 5**	**Top 2**	**No Caching**	**No Macro-Actions**
Keep All					
(p =)	0.002364	0.004143	0.8255	$5.897 * 10^{-05}$	$1.915 * 10^{-06}$
Sig?	Yes	Yes	No	Yes	Yes
Best	Top 10	Top 5		Keep All	Keep All
Top 10					
(p =)		0.8688	0.2704	$5.69 * 10^{-05}$	$8.813 * 10^{-08}$
Sig?		No	No	Yes	Yes
Best				Top 10	Top 10
Top 5					
(p =)			0.09114	$1.21 * 10^{-05}$	$3.564 * 10^{-08}$
Sig?			No	Yes	Yes
Best				Top 5	Top 5
Top 2					
(p =)				$5.219 * 10^{-05}$	$1.13 * 10^{-06}$
Sig?				Yes	Yes
Best				Top 2	Top 2
No Caching					
(p =)					$3.175 * 10^{-06}$
Sig?					Yes
Best					No Caching

Table 5.15: Significance table for the survival of the fittest strategy: p is the probability that the null hypothesis cannot be rejected; sig? denotes whether or not the null hypothesis, that the versions take the same time to solve problems, can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

that the number of times a macro-action has used is a good indicator of whether it is likely to be useful in the future: it is not simply the case that keeping any 10 macro-actions can lead to such good performance. It is, however, worth noting that the random version of the planner still performs better than the no macro-actions version, although not as well as the no caching version[5], this is a tribute to the effectiveness of the search time pruning. Another reason for the, perhaps unexpected, performance of this random version is that in two domains where macro-actions are very helpful—Philosophers, Briefcase—fewer than 10 macro-actions are generated, so there is no performance loss in those. Further, only 15 macro-actions are generated in the Satellite domain, and this is spread over a number of problems, meaning that the probability of the planner pruning the

[5]Note that the random version of the planner is still benefitting from the macro-actions generated on each individual problem instance

most useful macro-actions is low.

Something that has become clear during the analysis of the results in this section is that the main danger when keeping macro-actions is not the increase in branching factor, the search time pruning reduces this overhead greatly; it is, instead, the risk of learning some 'bad' macro-actions that lead the planner in a different direction in the search space causing it to fail to solve problems that it would otherwise have solved. Macro-actions that hinder performance are occasionally learnt and it is unclear how to identify them in an online manner: there is no way of knowing whether the problem would have been solved more quickly, or indeed at all, without attempting to solve it without that macro-action. Despite the occurrence of this phenomenon the planner does solve more problems overall, and more quickly on average. The phenomenon is just a different form of the trade-off introduced when using macro-actions: instead of the main potential problem being increasing the branching factor; it is the potential for learning 'bad' macro-actions. As with the concerns about increasing the branching factor, however, the overall benefits of using macro-actions far outweigh the costs associated with learning 'bad' macro-actions.

5.3.3 Time-Out Pruning

The hypothesis for this experiment is motivated as follows. Pruning macro-actions that have not been recently used will reduce the overheads incurred by reasoning about a large number of macro-actions. The macro-actions pruned will be those that have not been shown to be recently useful, hence useful macro-actions will not be too readily removed from the library. Timing out macro-actions with a time interval that is too short will cause the planner to delete macro-actions that improve search performance, thus giving worse performance; whereas a timeout interval that is too long will allow the size of the macro-action library to grow too large, resulting in the overheads of managing many macro-actions outweighing the benefits of using them. Time out intervals are measured based on the number of problems solved since a given macro-action was last used.

Hypothesis

Pruning macro-actions that have not been recently used will improve planner performance: maintaining useful actions; whilst reducing the branching factor by removing those that are not as useful.

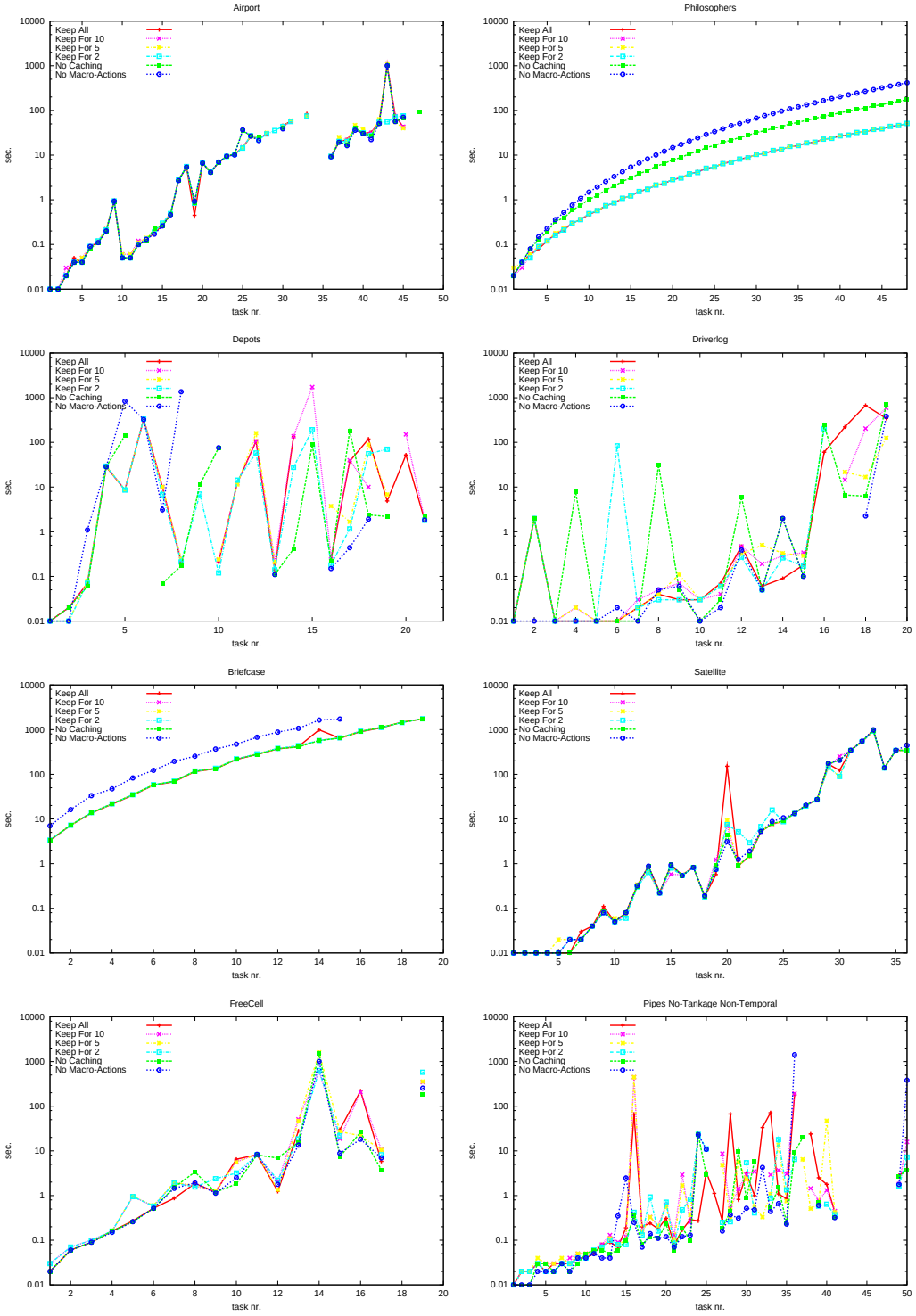


Figure 5.16: Time taken to solve problems in the evaluation domains using different configurations of the time-out pruning strategy (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	Keep All	Keep For 10	Keep For 5	Keep For 2	Keep For 1	No Cach-ing	No Macro-Actions
FreeCell	16	18	18	17	18	18	18
Airport	41	41	41	42	42	39	38
Depots	19	19	17	20	19	17	14
Philosophers	48	48	48	48	48	48	48
Driverlog	19	18	18	16	18	19	17
Pipes NT	42	39	39	38	41	39	39
Briefcase	19	19	19	19	19	19	15
satellite	36	36	36	36	36	36	36
Totals	**240**	**238**	**236**	**236**	**241**	**235**	**225**

Table 5.16: Coverage Across Evaluation Domains Varying Using the Time-Out Pruning Strategy

Analysis

Figure 5.16 shows the time taken by each of the time-out pruning versions of the planner; and that taken by the other control versions. The coverage achieved by all versions of the planner is summarized in table 5.16. In this experiment only one configuration solved more problems overall than the keep all version of the planner; this was the keep for 1 version solving one more problem. The caching strategies are, however, still interesting as much of this difference in coverage occurs in one domain, Pipes No-Tankage; discounting that domain the other strategies are comparable. In some domains the other strategies perform better than keeping all macro-actions. Furthermore, several other versions of the planner produce plans superior in quality to those produced by the keep all strategy.

It may, at first, appear confusing that figure 5.18 shows that some of the configurations of the planner pruning more aggressively actually have greater macro-action library sizes, at some points, than those keeping actions for fewer problems. This is, however, not a problem with the pruning strategy, merely an artifact of different libraries of macro-actions leading search in different directions. Suppose the pruning strategy managing library A is keeping macro-actions for a shorter time than that managing library B, and after solving some problem p the library sizes are 3 and 5 respectively. When solving problem $p + 1$ the different macro-action libraries may lead search in different directions: an action from library B may be selected to solve the problem with library B; while the planner using library A must proceed in a different direction. It is quite possible, therefore that the version using library A will encounter more plateaux during

search, and thus generate more macro-actions, than that using library B. If, in this example, A encountered 10 previously unseen plateaux and B only 6 then the macro-action library size of A would become 13 and that of B would only be 11.

As in the survival of the fittest strategy, the results for the Airport domain are quite similar for all configurations as many problems are solved by best-first search (see section 5.3.2). Each of the caching strategies solves more problems than the control version through avoiding resorting to best-first search, as in the survival of the fittest strategy. Again very little difference is observed in the makespan of the plans produced by each of the versions. The macro-action library sizes for the time-out pruning strategy, shown in figure 5.18 are more varied those for the time-out pruning strategy; this is because there is no upper bound on library size. Indeed it can be seen that even the most aggressive pruning strategy, only allowing macro-actions to remain in the library for one problem if they are not used, has a peak library size of 19. This is in contrast to the peak library size of 2 found in the survival of the fittest strategy keeping the top 2 actions. In fact, in the Airport domain the most aggressive of the time-out pruning strategies is able to solve more problems that the most aggressive of the survival of the fittest strategies. The best performing of the survival of the fittest configurations—top 10, solving 44 problems—is, however, not matched by any of the time-out pruning versions.

The performance in the Philosophers domain is similar to that observed when using the survival of the fittest caching strategy. Recall that in this domain only three distinct plateau types are met after solving the first two problems. The macro-action libraries generated in this domain better reflect this than those generated in the survival of the fittest strategy. Figure 5.18 shows that all configurations of the time-out pruning strategy converge to the library size of three after $k+2$ problems, where k is the number of problems for which unused macro-actions are kept. The macro-action library generated by the time-out strategy in this domain is, therefore, more compact than those generated by the survival of the fittest strategy, without losing any useful macro-actions. There is, however, not a dramatic improvement in performance seen between using this compact library and that generated by the keep all version of the planner. This is because the macro-actions are ordered according to their usage counts: since the only three macro-actions that are ever used are the top three in the library, based on usage count, the other macro-actions in the library are never actually considered. The structure of this domain is such that when the planner reaches a plateau, one

Domain	Keep All	Keep For 10	Keep For 5	Keep For 2	Keep For 1	No Caching
FreeCell	-14.91 *(16)*	2.94 *(18)*	-42.35 *(18)*	1.97 *(17)*	22.67 *(18)*	-25.90 *(18)*
airport	-0.61 *(38)*	-3.18 *(38)*	-2.57 *(38)*	24.26 *(38)*	-1.69 *(38)*	-0.47 *(38)*
depots	147.82 *(14)*	157.19 *(14)*	154 *(14)*	157.01 *(14)*	157.66 *(14)*	142.12 *(13)*
philosophers	77.43 *(48)*	77.45 *(48)*	77.43 *(48)*	77.43 *(48)*	77.44 *(48)*	50.00 *(48)*
driverlog	-36.98 *(17)*	-24.32 *(17)*	14.29 *(17)*	-5.56 *(15)*	-58.67 *(17)*	-21.89 *(17)*
Pipes NT	35.97 *(39)*	30.31 *(37)*	-13.07 *(36)*	47.67 *(37)*	45.95 *(38)*	46.89 *(38)*
Briefcase	281.14 *(15)*	306.19 *(15)*	305.86 *(15)*	304.82 *(15)*	304.54 *(15)*	308.74 *(15)*
satellite	2.23 *(36)*	5.44 *(36)*	9.93 *(36)*	8.98 *(36)*	9.41 *(36)*	3.59 *(36)*
Totals	**61.51** ***(223)***	**69** ***(223)***	**62.94** ***(222)***	**77.07** ***(220)***	**69.66** ***(224)***	**62.88** ***(223)***

Table 5.17: Mean of time taken by no macro-actions version minus time taken using each configuration of the time-out pruning strategy. Results are calculated on mutually solved problems

of the best 3 macro-actions will lead to a better state, and will thus be chosen before the other macro-actions are considered.

When using the survival of the fittest strategy in the Depots domain macro-actions were observed to be very helpful; a fact that can also be observed using this caching strategy. Indeed this pruning strategy performs as well as the survival of the fittest pruning strategy with versions pruning macro-actions able to solve at least as many problems as those using the survival of the fittest pruning strategy. The benefits gained by some versions of the planner using this strategy, over versions keeping the top n macro-actions, are because of the many different types of plateaux encountered. A more diverse range of plateaux occur in the Depots domain than in other domains for which macro-actions are useful, such as Philosophers and Briefcase, and allowing macro-actions to persist if they have been used even once recently can allow the planner to solve more problems. The lack of an imposed upper bound on the size of the library means more macro-actions can be retained if they appear useful.

The performance in the Briefcase domain is similar to that of the survival of

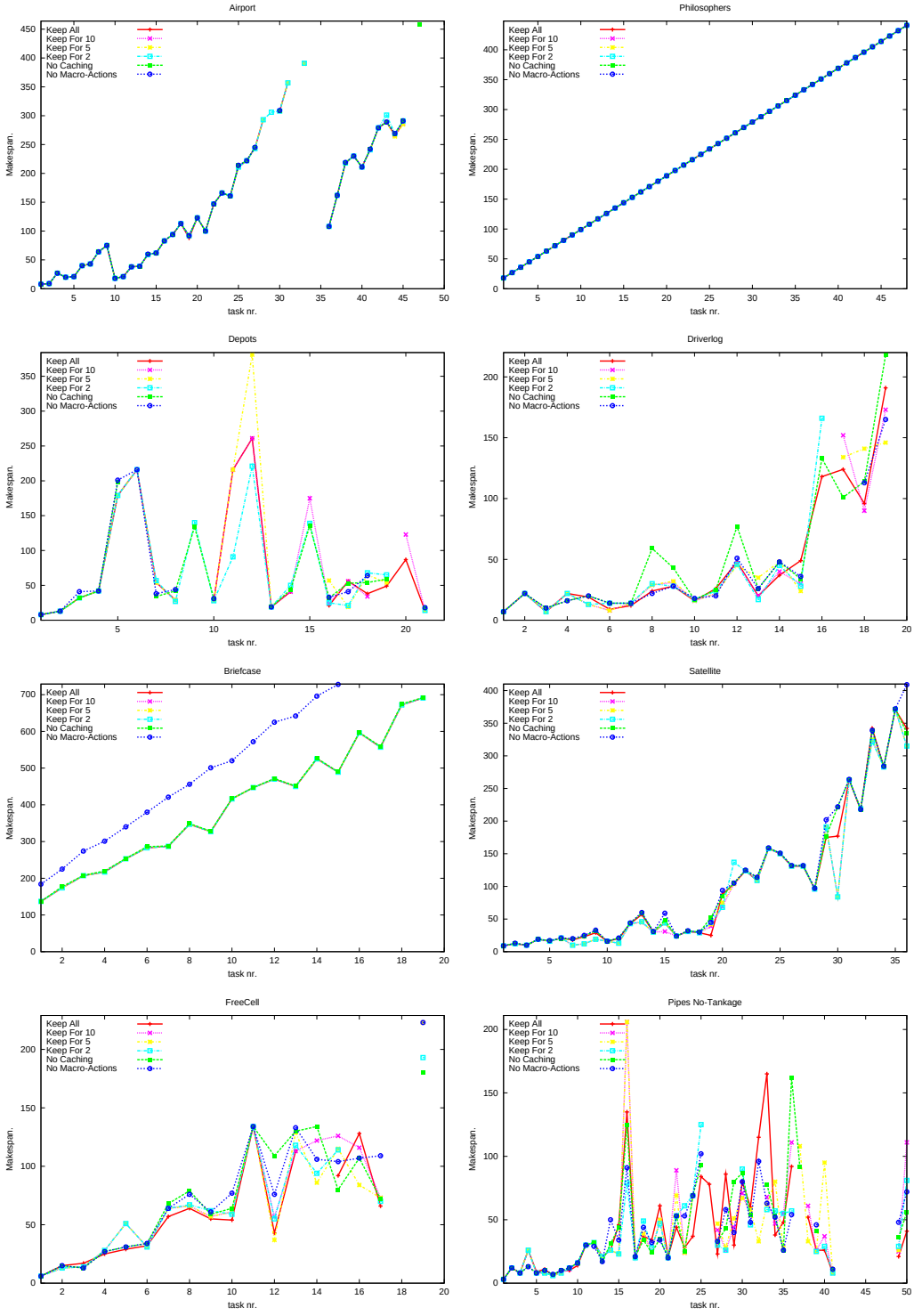


Figure 5.17: Makespan of plans generated in the evaluation domains using different configurations of the time-out pruning strategy (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

the fittest strategy. Only two macro-actions are generated and each of these is used at least once on all problems; there is therefore no difference in the macro-action libraries kept by each of the strategies, and hence no difference in performance.

In the Satellite domain the same phenomenon occurs as in the survival of the fittest strategy: an overall slight time improvement is observed as shown in table 5.17. The improvement is slight because, although plateau-escaping sequences can be reused, the plateaux encountered are small and few in number. In this domain in the survival of the fittest configurations keeping smaller numbers of macro-actions were generally more successful. Figure 5.18 shows that the keep for 5 version has a small macro-action library containing 5 or fewer macro-actions for all except 5 problems. This version is the most successful of all the pruning methods so far in the Satellite domain solving problems, on average, the fastest. The library size adapts appropriately to the domain: it remains small for most problems but there is a spike in problems 19 and 20 when many plateaux are encountered. The macro-actions used here are found not to be useful and pruned reasonably quickly so that the library does not grow too large or remain large for a long time. The keep for 1 and keep for 2 versions also exhibit adaptive behaviour and perform almost as well; they do, however, disregard macro-actions slightly too quickly thus not gaining quite the same benefits as the keep for 5 strategy. The keep for 10 and keep all strategies become slightly overladen with macro-actions that are not quite so useful and may guide search in the wrong direction, or cause the search to progress more slowly. The macro-actions are very effective at avoiding exhaustive search in this domain: the version of the planner using no macro-actions has to use exhaustive search during EHC on 33 of the 36 problems; the version keeping all macro-actions only uses exhaustive search on 7 of the 36 problems. The keep for 5 version uses exhaustive search on 8 of the problems but is still faster than the keep all version as it does not have the overheads of too many macro-actions.

Conclusion

A library of macro-actions, managed using the time-out pruning strategy, improves on the performance of the no macro-actions and no caching versions of the planner across the evaluation domains. The number of problems solved increases and the problems are solved, on average, over 60 seconds more quickly. Only one of configurations solve as many problems overall as the version keeping all macro-actions but others do solve more problems in some domains and can

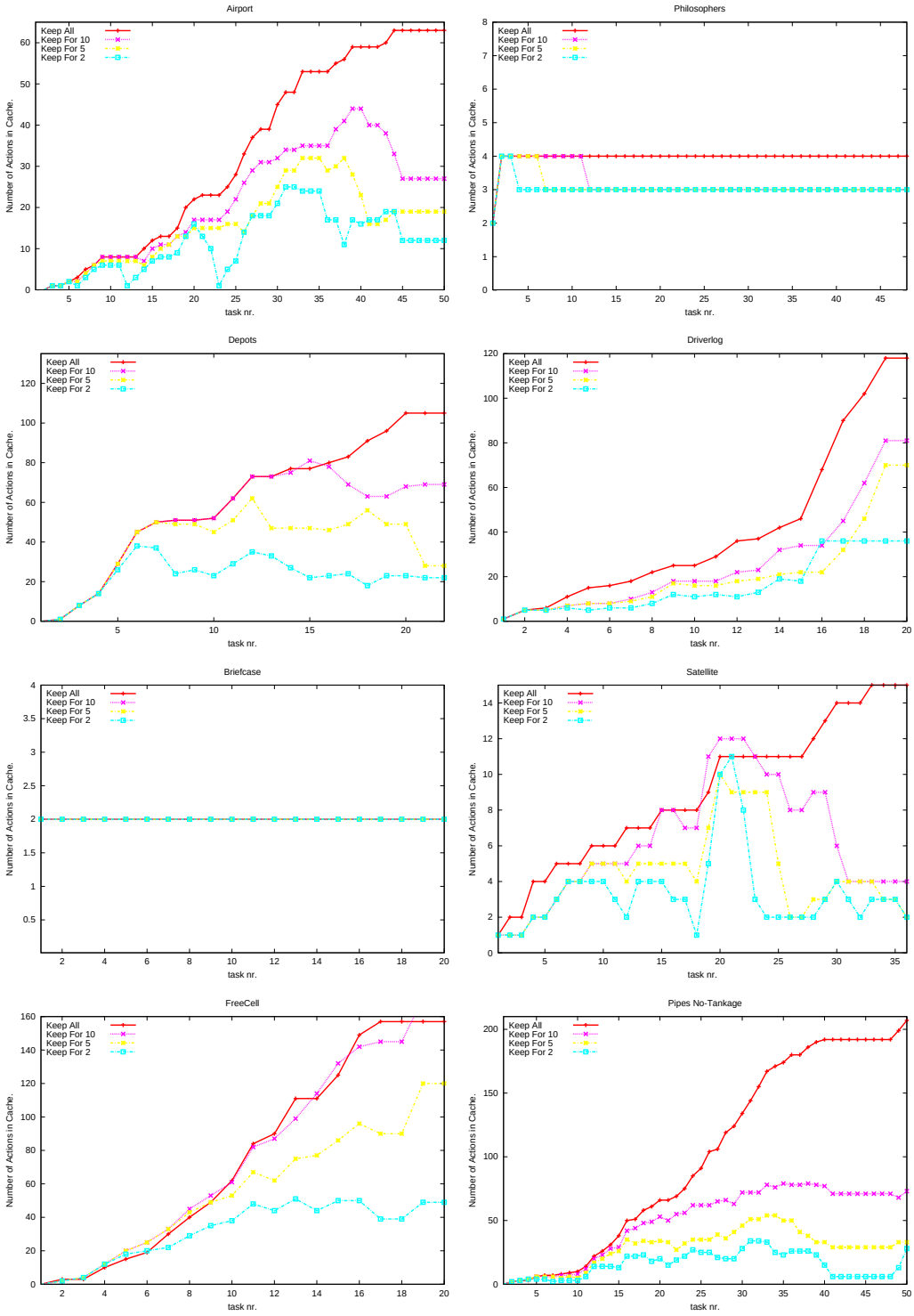


Figure 5.18: Length of the macro-action library generated in the evaluation domains varying the time-out cut off (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

	Keep For 10	**Keep For 5**	**Keep For 2**	**Keep For 1**	**No Caching**	**No Macro-Actions**
Keep All						
(p =)	0.01466	0.08425	0.7484	0.9578	$5.897 * 10^{-05}$	$1.915 * 10^{-06}$
Sig?	Yes	No	No	No	Yes	Yes
Best	Keep All				Keep All	Keep All
Keep For 10						
(p =)		0.5257	0.08263	0.4079	0.1267	0.0004913
Sig?		No	No	No	No	Yes
Best						Keep For 10
Keep For 5						
(p =)			0.02184	0.1127	0.0307	$9.406 * 10^{-05}$
Sig?			Yes	No	Yes	Yes
Best			Keep For 2		Keep For 5	Keep For 5
Keep For 2						
(p =)				0.7307	0.005629	$3.741 * 10^{-06}$
Sig?				No	Yes	Yes
Best					Keep For 2	Keep For 2
Keep For 1						
(p =)					0.0111	$3.203 * 10^{-06}$
Sig?					Yes	Yes
Best					Keep For 1	Keep For 1
No Caching						
(p =)						$3.175 * 10^{-06}$
Sig?						Yes
Best						No Caching

Table 5.18: Significance table for the time out pruning strategy: p is the probability that the null hypothesis, that the versions take the same time to solve problems, cannot be rejected; sig? denotes whether or not the null hypothesis can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

solve problems more quickly on average. As discussed in section 5.3.2 the good performance of the keep all version can be attributed to the effective search-time pruning employed.

It is clear from figure 5.19 that the two best performing versions of the planner, in terms of number of problems solved, are the keep all version and the keep for 1 version; indeed at some time cut off points the number of problems solved by these is identical. The versions in between these do not perform as well. At one end of the scale the keep all version of the planner is benefiting from not having to prune potentially useful macro-actions, being guided in which ones to use by the search time pruning. In contrast to this the keep 1 version of the planner is maintaining a significantly smaller macro-action library and is thus gaining the

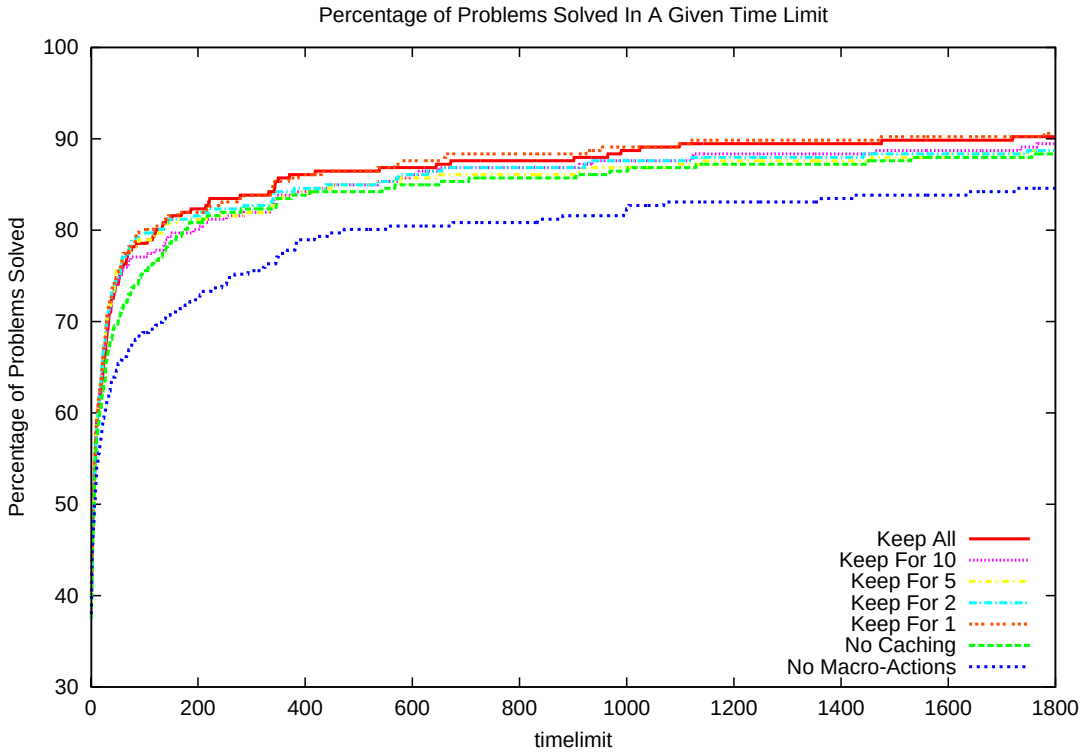


Figure 5.19: Percentage of Problems Solved in a Given Time Limit Using the Time-Out Pruning Strategy

benefits of not having to reason about large numbers of macro-actions, which was shown to offer slight benefits when keeping the top 10 macro-actions outperformed keeping all macro-actions under the survival of the fittest strategy. The average macro-action library size for the keep for 1 version of the planner is 9.4; whereas for the keep all version of the planner it is over four times the size at 42.2. The strategies between these are losing both the benefit of small library sizes and that of not discarding potentially useful macro-actions and therefore do not perform as well.

Table 5.18 shows the results of performing Wilcoxon signed-rank tests to determine whether there is significant difference in the time taken to solve problems by each of the versions. It is difficult to show a significant difference in performance between the different configurations of the time-out pruning strategy indicating that the varying time-out does not have a great impact on time taken to solve problems in the chosen domains. All versions keep a reasonably large library; the main difference seen is in terms of coverage which is, of course, not reflected in this test; the test only consideres mutually solved problems.

The makespan of plans is again improved when using this macro-action caching strategy, all of the versions produce plans that are, on average, approxi-

mately 17 time steps shorter than those produced using the no macro-actions version of the planner, again showing that macro-actions can improve the makespan of plans generated.

5.3.4 Roulette Selection

Hypothesis

The roulette selection strategy will perform similarly to the survival of the fittest strategy but will be more successful for domains in which the top n macro-actions are not necessarily the best macro-actions to use all of the time. The bias will allow the planner to select macro-actions that appear useful without constraining the planner to always necessarily select the same macro-actions for use.

Analysis

The performance in the Airport domain using this strategy (shown in figure 5.20) is not quite as good as that using the survival of the fittest strategy. As discussed in section 5.3.2 the performance of the planner is not improved greatly by macro-actions because many of the problems are solved by best first search. One observable difference is the loss of the benefits seen on problem 43 that the macro-action caching versions achieved in the survival of the fittest strategy. In this problem using the roulette strategies none of the versions of the planner select the necessary collection of macro-actions that are used to achieve this performance improvement. Further in the range 45-50 many of the problems that were solved by the top n macro-action caching versions are not solved by the roulette selection versions. In not always selecting the most used macro-actions in this domain the planner is not receiving search guidance that is as useful. Instead EHC fails and the planner using these strategies resorts to best-first search in solving many problems, just as the no macro-actions version does.

The performance in the Philosophers domain is similar, for most configurations, to that in the survival of the fittest strategy. When the planner is selecting more than three macro-actions the best performance, as seen in the survival of the fittest strategy, is achieved. This is because there are three useful macro-actions in this domain and all are guaranteed be selected from the overall collection of 4 by any planner version selecting at least 4 macro-actions. The performance of the versions selecting one or two macro-actions is, however, somewhat different. The version selecting two macro-actions generates a different macro-action library containing 6 macro-actions. The different macro-action library is generated when

Figure 5.20: Time taken to solve problems in the evaluation domains using roulette selection (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	Keep All	Sel. 10	Sel. 5	Sel. 2	Sel. 1	No Caching	No Macro-Actions
FreeCell	16	19	19	19	18	18	18
airport	41	42	40	40	39	39	38
depots	21	19	19	19	16	17	14
philosophers	48	48	48	48	14	48	48
driverlog	19	18	17	17	17	19	17
Pipes NT	42	41	42	41	39	39	39
Briefcase	19	18	18	18	12	19	15
satellite	36	36	34	35	35	36	36
Totals	**242**	**241**	**237**	**237**	**190**	**235**	**225**

Table 5.19: Coverage Across Evaluation Domains Using the Roulette Selection Strategy

search proceeds in a different direction due to the selection of a combination of macro-actions. Selecting two actions from this library still allows the planner to perform better than the no caching version. The select 1 version, however, learns a collection of macro-actions from which the selection of most single macro-actions causes the planner to resort to best-first search. Best-First search is unsuccessful in this domain for solving large problems so the performance of this version of the planner is severely degraded.

Macro-actions remain a successful technique when used in the Depots domain. As shown in table 5.19, Selecting 2, 5 or 10 macro-actions performs well allowing the planner to solve 19 of the 22 problems in this domain. The Depots domain consists of a large number of different plateaux: in this domain it is not always the most used macro-actions that are the best to consider for re-use. A diverse range of plateaux are encountered and selecting a diverse range of macro-actions, therefore, assists the planner in finding a solution to the problem. In terms of solution quality in general the more macro-actions that are selected, the better the quality of the resulting plan. Macro-actions are known to have the potential to decrease plan length in this domain, the more that are available to the planner the greater the chance of finding a macro-action that leads to a shorter route off a plateau.

The spikes caused by resorting to best first search when solving the easier problems in the Driverlog domain occur less frequently in the roulette selection strategy. The planner often avoids selecting a problematic macro-action which will lead it into a dead end. The no caching version still resorts to best-first search on some easier problem instances but the use of roulette selection eliminates this

Domain	**Keep All**	**Sel. 10**	**Sel. 5**	**Sel. 2**	**Sel. 1**	**No Caching**
FreeCell	-14.91 *(16)*	-7.13 *(18)*	-9.13 *(18)*	10.03 *(18)*	53.29 *(18)*	-25.90 *(18)*
airport	-0.61 *(38)*	2.23 *(38)*	1.33 *(38)*	0.85 *(38)*	1.10 *(38)*	-0.47 *(38)*
depots	147.82 *(14)*	158.8 *(14)*	142.14 *(14)*	62.86 *(13)*	62.96 *(13)*	142.12 *(13)*
philosophers	77.43 *(48)*	64.23 *(48)*	64.26 *(48)*	51.63 *(48)*	-3.79 *(14)*	50.00 *(48)*
driverlog	-36.98 *(17)*	-59.53 *(16)*	0.01 *(15)*	-5.04 *(16)*	-6.16 *(16)*	-21.89 *(17)*
Pipes NT	35.97 *(39)*	46.64 *(37)*	45.04 *(39)*	0.72 *(38)*	0.97 *(37)*	46.89 *(38)*
Briefcase	281.14 *(15)*	301.54 *(15)*	301.47 *(15)*	301.42 *(15)*	-108.29 *(12)*	308.74 *(15)*
satellite	2.23 *(36)*	5.09 *(36)*	-26.89 *(34)*	2.51 *(35)*	2.03 *(35)*	3.59 *(36)*
Totals	**61.51** ***(223)***	**63.98** ***(222)***	**64.77** ***(221)***	**53.12** ***(221)***	**0.26** ***(183)***	**62.88** ***(223)***

Table 5.20: Mean of time taken by no macro-actions version minus time taken using each configuration of the roulette selection strategy. Results are calculated on mutually solved problems

behaviour on all problems except problem 2. On problem 2 only one macro-action has been generated thus selecting any macro-actions from the library will necessarily result in the same performance as the survival of the fittest strategy. On subsequent problems more macro-actions have been generated, the planner does not select the problematic action and does not encounter the dead end. The performance of the roulette selection strategies on the harder problems in this domain is, however, not as good as that of the survival of the fittest strategy. On the harder problems the versions using survival of the fittest strategy have attempted to use all of the best macro-actions on all of the problems so far and maintained a good ranking of these. The rankings obtained by the roulette selection strategy will depend on which macro-actions have been selected for use, and therefore may not be as accurate. The survival of the fittest strategy is therefore more successful on the harder problems as it has learnt a more useful set of macro-actions.

The results in the Briefcase domain are the same as the survival of the fittest results for most configurations: at most two macro-actions are generated across

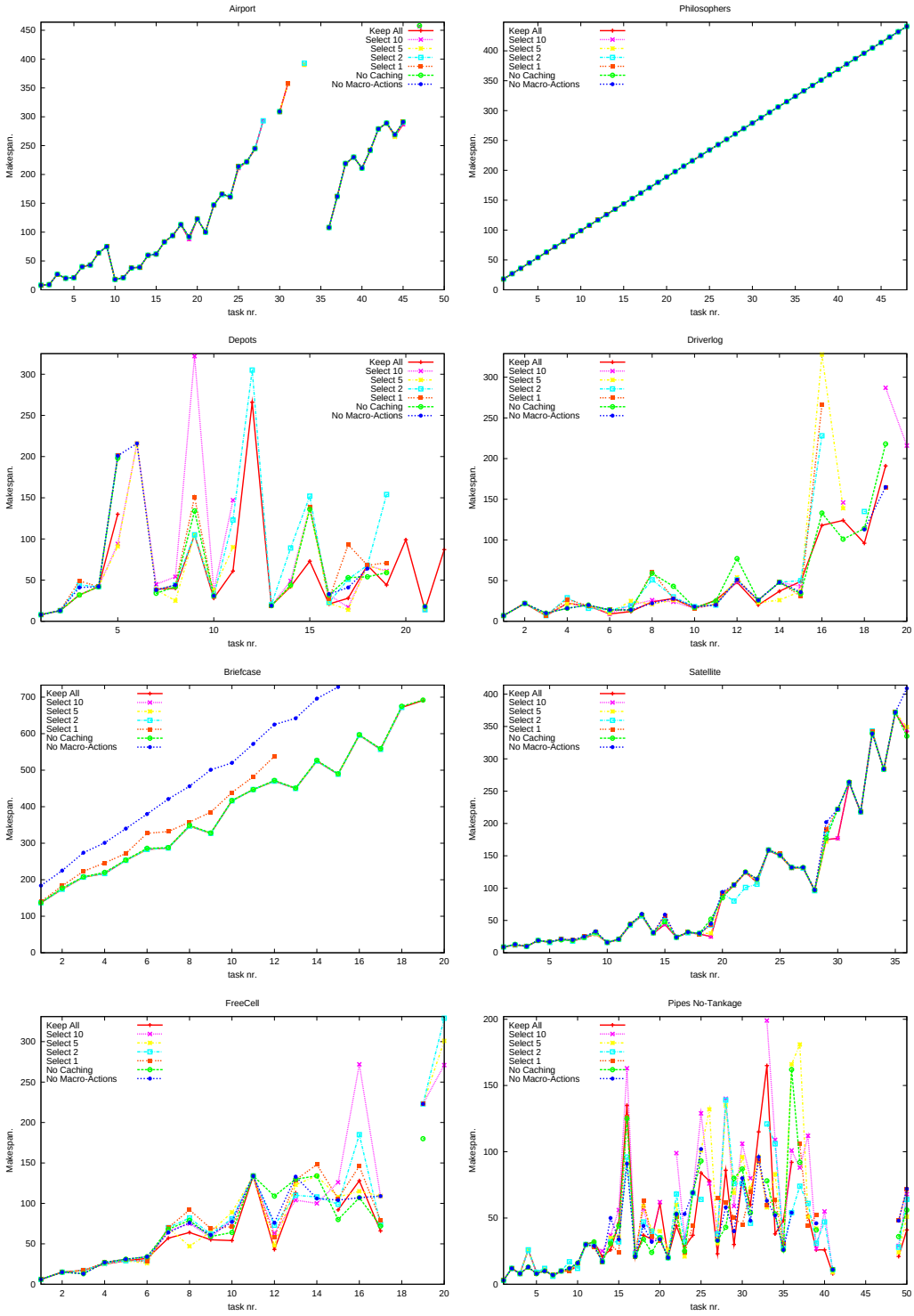


Figure 5.21: Makespan of plans generated in the evaluation domains using roulette selection (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

the entire problem suite so all of the selection strategies, except select 1, perform as the keep all version. The select 1 strategy does not perform as well as the other roulette selection strategies in this domain; nor as well as the no caching version. This is because searching with only one of the two macro-actions leads search on a different trajectory through the EHC landscape. Search never encounters the other plateau type until the very last two actions of the plan; instead the one chosen macro-action is used to bypass all plateaux thus creating a different plan. The different path taken still leads to shorter plans than those generated using no macro-actions; but longer plans than those generated by the other macro-action using versions of the planner.

In the Satellite domain the effects of using macro-actions are limited by the relatively small number of plateaux. Slight overall performance improvements can be seen, in table 5.20, using each of the macro-action caching strategies (except for the select 5 strategy). In the select 5 strategy, however, the planner is unable to solve two problems that are solved by all other configurations. In problem 30 the planner reaches a state in which there are many helpful actions and the planner spends the remainder of the time exhausting these; the choice of macro-actions in problem 32 leads the planner to a dead end and forces it to resort to best first search. Roulette selection of actions in this manner can lead to choosing collections of macro-actions that have not been tested as a single combined unit on previous problems and, when used together, can make solving the problem more difficult.

Conclusion

Roulette selection of macro-actions from a library can significantly improve performance and coverage in a number of domains compared to not using macro-actions. Table 5.3.4 shows that all versions using roulette selection, except the select 1 version, solve problems significantly more quickly than the no macro-actions an no caching versions. Further, almost every strategy selecting more macro-actions than the strategy to which it is being compared solves mutually solved problems significantly more quickly. This indicates that the roulette selection does not necessarily select good actions from the collection available, and the more actions that are selected, the more chance a useful one is selected. The hypothesis that the roulette selection strategy performs similarly to the survival of the fittest strategy, and in some cases improves upon it, does not hold.

The summary graph in figure 5.22 shows that none of the versions using roulette selection are able to solve more problems than the keep all version of the

	Sel. 10	**Sel. 5**	**Sel. 2**	**Sel. 1**	**No Caching**	**No Macro-Actions**
Keep All						
(p =)	0.05668	0.01256	0.002414	0.8655	$5.897 * 10^{-05}$	$1.915 * 10^{-06}$
Sig?	No	Yes	Yes	No	Yes	Yes
Best		Keep All	Keep All		Keep All	Keep All
Sel. 10						
(p =)		0.1186	0.00108	0.02367	0.0003103	$4.464 * 10^{-11}$
Sig?		No	Yes	Yes	Yes	Yes
Best			Select 10	Select 10	Select 10	Select 10
Sel. 5						
(p =)			0.001657	0.09429	0.0004504	$1.5 * 10^{-10}$
Sig?			Yes	No	Yes	Yes
Best			Select 5		Select 5	Select 5
Sel. 2						
(p =)				0.02543	0.001613	$2.012 * 10^{-11}$
Sig?				Yes	Yes	Yes
Best				Select 2	Select 2	Select 2
Sel. 1						
(p =)					0.5872	0.2401
Sig?					No	No
Best						
No Caching						
(p =)						$3.175 * 10^{-06}$
Sig?						Yes
Best						No Caching

Table 5.21: Significance table for the roulette selection strategy: p is the probability that the null hypothesis, that the versions solve problems in the same length of time, cannot be rejected; sig? denotes whether or not the null hypothesis, that the versions perform the same, can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

planner. Further the results are not as good as those generated using the survival of the fittest strategy. Selecting actions deterministically gives rise to better performance: this implies that past use count is, in fact, a good indicator of the best macro-actions to consider, and that considering other actions instead some of the time is not helpful in general. The performance using roulette selection is, in general, more sporadic than that using the survival of the fittest strategy and varies depending on which macro-actions are selected from the library. An improvement in makespan, shown in figure 5.21, is seen when using the roulette selection strategy. This is, however, not quite as great as that seen using the equivalent survival of the fittest configurations.

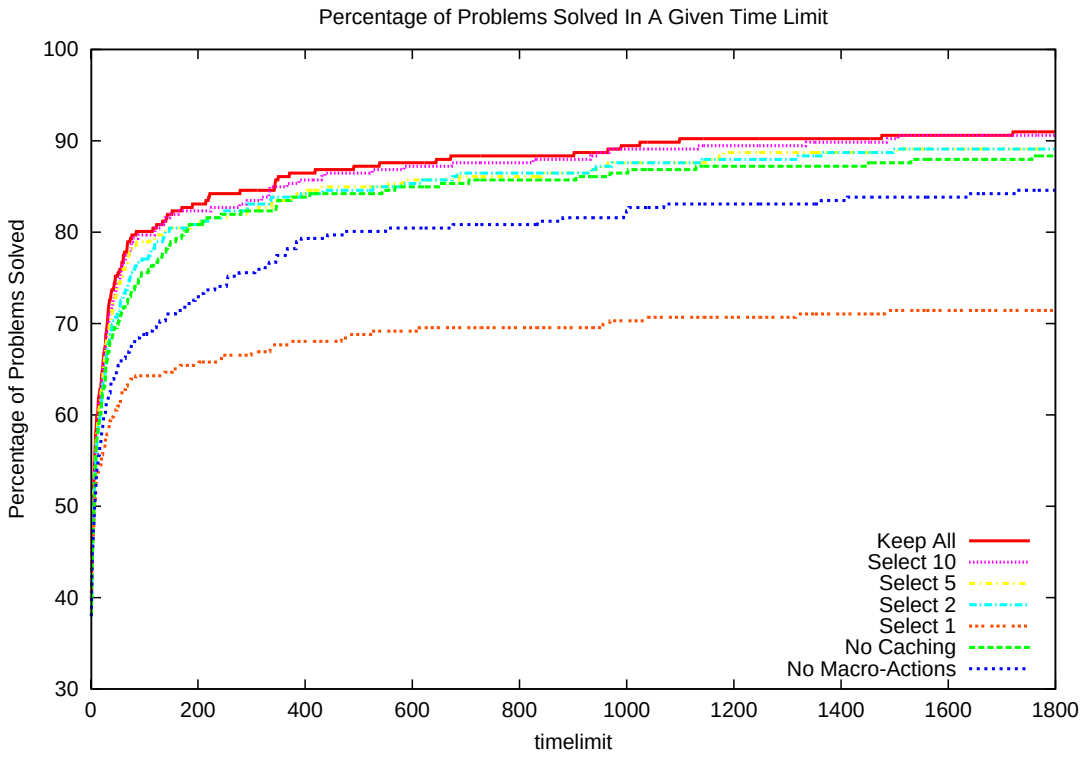


Figure 5.22: Percentage of Problems Solved in a Given Time Limit Using Roulette Selection from the Macro-Action Library

5.3.5 Pruning based on Instantiation Versus Usefulness

The use of macro-actions in planning introduces a trade-off. The branching factor of the search is increased as more actions must be considered at each choice point; the potential depth to which the search space must be explored is, however, reduced if the correct macro-actions are selected. The cost of keeping and using a macro-action depends on the number of times it is instantiated; the benefits, in decreasing the depth to which the search space must be explored, depend on how many times the macro-action is used in a solution plan[6]. Pruning based on the ratio of instantiation versus use count (dividing the number of times a macro-action has been used by the number of times it has been instantiated) should, therefore, allow the planner to keep the macro-actions that offer the best value; that is, those that cost the least to maintain relative to how useful they are. Keeping only the macro-actions that are the least expensive for the benefit they give should give rise to a performance improvement.

[6]A macro-action does not necessarily create a shorter plan if it is used but we have seen in section 5.2.1 that this is often the case; also avoiding exhaustive search on plateaux through the use of a macro-action is likely to make search faster (see section 5.3.2). Further if a macro-action plans n steps at once this reduces the depth of search tree exploration to find the same plan by n-1.

Domain	Keep All	0.001	0.002	0.01	0.02	0.1	No Caching	No Macro-Actions
FreeCell	16	18	19	17	18	18	18	18
Airport	41	42	42	42	43	42	39	38
Depots	19	19	17	18	19	17	17	14
Philosophers	48	48	48	48	48	48	48	48
Driverlog	19	16	16	16	16	16	19	17
Pipes NT	42	40	41	41	39	42	39	39
Briefcase	19	18	18	19	19	19	19	15
Satellite	36	36	36	35	36	36	36	36
Totals	**240**	**237**	**237**	**236**	**238**	**238**	**235**	**225**

Table 5.22: Coverage Across Evaluation Domains Using Instantiation Versus Use Pruning

Hypothesis

Pruning macro-actions from the library based on the ratio between the number of times they have been instantiated and the number of times they have been used (keeping those that are both instantiated few times and used many times) will improve planner performance.

Analysis

The coverage in the Airport domain, shown in table 5.22, for all caching versions improves on that for both the no caching and keep all versions of the planner. All instantiation versus use pruning versions are maintaining a macro-action library of similar size, which is slightly smaller than that maintained by the keep all version. The peak coverage of all versions so far is seen in the 0.02 version of the planner which successfully manages to solve 43 of the 50 problems in this domain. All the versions caching macro-actions manage to achieve the benefits observed in the survival of the fittest strategy section (section 5.3.2) of solving more problems and resorting to best-first search on only 15 instead of 19 problems. The library sizes for the pruning versions, shown in figure 5.24, is slightly smaller than that for the keep all version; however, most of the macro-actions are kept by all of the versions, implying that macro-actions in the Airport domain are generally used at least one tenth of the times they are instantiated.

In the Philosophers domain the macro-action library is quickly pruned by all pruning versions to contain only the three most useful macro-actions. In fact, in this domain, the three most useful macro-actions all have an instantiation/use

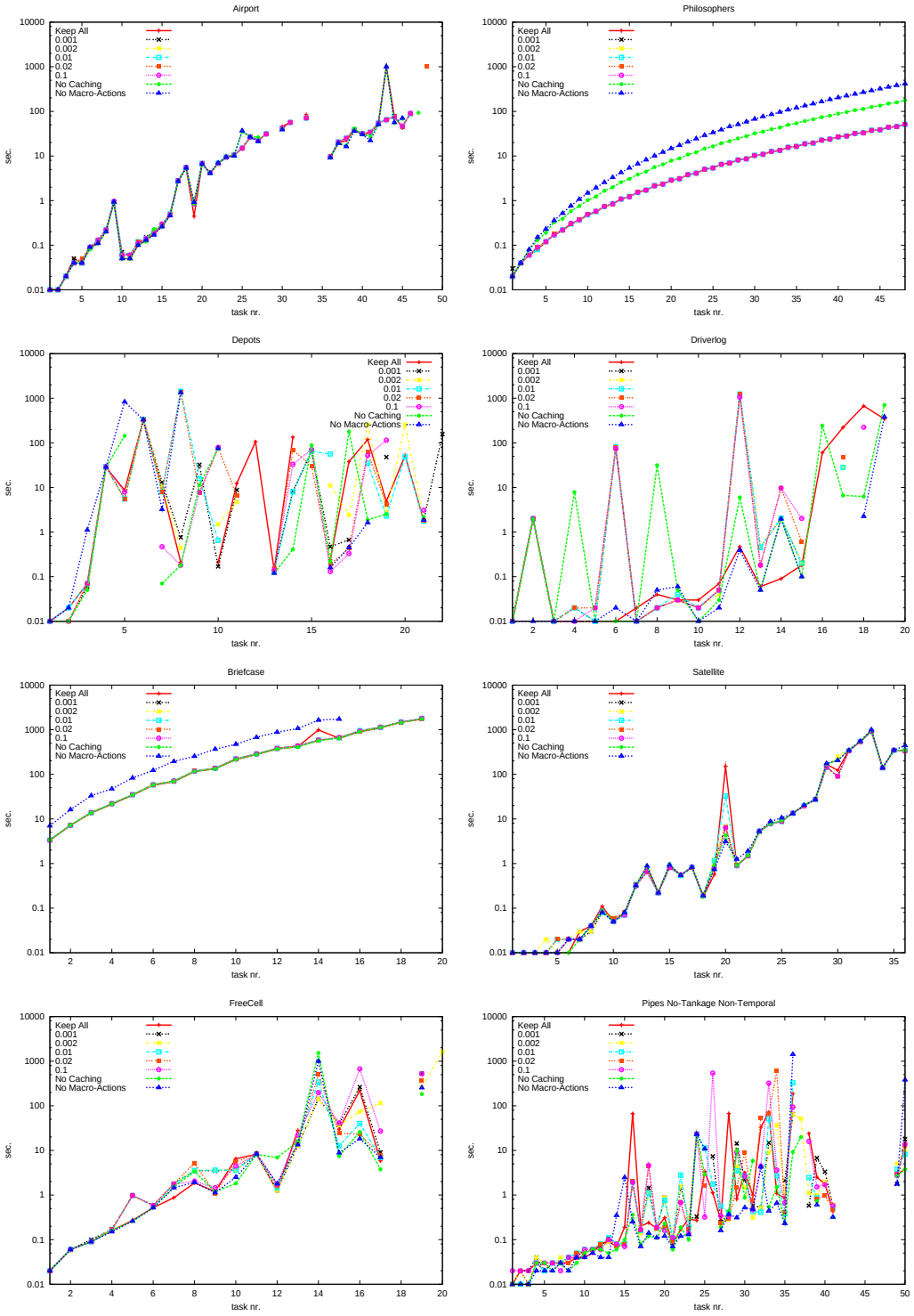


Figure 5.23: Time taken to solve problems in the evaluation domains using instantiation versus use pruning (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

ratio of approximately 1; that is, every time they are instantiated they are used. No obvious performance benefits are seen, however, over the keep all version because only four macro-actions are ever generated in this domain. Although the fourth one is pruned no great speed increase is seen because the last macro-action is never instantiated anyway even if not pruned. Pruning allows a smaller library to be maintained but has no impact on search performance in this domain. The comparison of the versions using macro-actions to those not is the same as in survival of the fittest strategy: the versions caching macro-actions have considerably better performance than the version not caching macro-actions; which in turn performs considerably better than the version using no macro-actions. This is a result of the planner avoiding the need to do exhaustive search by using macro-actions to step over similar plateaux (see section 5.3.2 for a more detailed explanation).

The keep all version is again very successful in solving problems in the Depots domain; the coverage it achieves is, however, equalled by the 0.001 and 0.02 versions of the planner. These versions both manage the same coverage with considerably smaller macro-action libraries. After solving the final problem the 0.001 and 0.02 versions of the planner have macro-action libraries of length 23 and 26 respectively; whereas the keep all version has a library of size of 105. The best performing two instantiation versus use caching versions are able to keep the useful macro-actions whilst pruning other irrelevant actions.

In the Driverlog domain the problems associated with resorting to best-first search when using macro-actions can still be seen as spikes in the corresponding graph in figure 5.23. These problems do, however, occur in fewer configurations, and less frequently, when using instantiation versus use pruning. In the earlier strategies it was observed that the versions of the planner keeping the most macro-actions were the only ones able to avoid this. Here, however, the instantiation versus use strategies prune the library quite aggressively, as shown in figure 5.24, keeping the keeping useful macro-actions whilst removing the necessary macro-actions to avoid resorting to best-first search.

In the Briefcase domain the library is actually lightly pruned by the 0.1 version, the version doing the most strict pruning. The macro-action that is used the second most is pruned after problem 10 is solved as it has been instantiated 1243 times but used only 163 times, which is not sufficient for it to have a ratio greater greater than 0.1. The macro-action reappears on every other problem when the post processing to identify macro-actions that appear in the plan is done and the use count is updated accordingly; of course the instantiation count

Domain	**Keep All**	**0.001**	**0.002**	**0.01**	**0.02**	**0.1**	**No Caching**
FreeCell	-14.91 *(16)*	16.95 *(18)*	36.55 *(18)*	36.98 *(17)*	19.96 *(18)*	-9.85 *(18)*	-25.90 *(18)*
Airport	-0.61 *(38)*	24.36 *(38)*	24.35 *(38)*	24.34 *(38)*	24.37 *(38)*	24.40 *(38)*	-0.47 *(38)*
Depots	147.82 *(14)*	170.91 *(13)*	139.52 *(14)*	53.92 *(13)*	49.55 *(14)*	161.92 *(13)*	142.12 *(13)*
Philosophers	77.43 *(48)*	77.45 *(48)*	77.45 *(48)*	77.44 *(48)*	77.44 *(48)*	77.45 *(48)*	50.00 *(48)*
Driverlog	-36.98 *(17)*	-87.63 *(15)*	-87.46 *(15)*	-87.74 *(15)*	-88.52 *(15)*	-85.1 *(16)*	-21.89 *(17)*
Pipes NT	35.97 *(39)*	46.06 *(37)*	44.05 *(38)*	36.01 *(39)*	-9.45 *(38)*	34.91 *(39)*	46.89 *(38)*
Briefcase	281.14 *(15)*	303.41 *(15)*	303.3 *(15)*	303.94 *(15)*	304.38 *(15)*	302.41 *(15)*	308.74 *(15)*
Satellite	2.23 *(36)*	9.56 *(36)*	4.87 *(36)*	5.81 *(35)*	9.66 *(36)*	9.4 *(36)*	3.59 *(36)*
Totals	**61.51** ***(223)***	**70.13** ***(220)***	**67.83** ***(222)***	**56.33** ***(220)***	**48.42** ***(222)***	**64.44** ***(223)***	**62.88** ***(223)***

Table 5.23: Mean of time taken by no macro-actions version minus time taken using each configuration of the instantiation versus use pruning strategy. Results are calculated on mutually solved problems

remains the same since the macro-action has not been included in search and thus not instantiated. When the macro-action returns to the library the instantiation count again becomes more than 10 times the use count and the macro-action is thus pruned. As observed in the survival of the fittest section (section 5.3.2) pruning the macro-actions from the library has little effect on performance. The macro-actions are quickly generated at the start of search as the plateaux are short. The only time at which the difference in using macro-actions and not using macro-actions is clearly visible in this domain is across the solution of a whole problem not using macro-actions where the number of plateaux avoided becomes large.

The performance in the Satellite domain using instantiation versus use pruning is comparable to the best seen so far. Coverage, as usual, is identical with the planner able to solve all problems in all but one of the versions. The time taken to solve problems is, however, on average in excess of 9 seconds faster than that when using other caching strategies, as shown in table 5.23. This performance is only

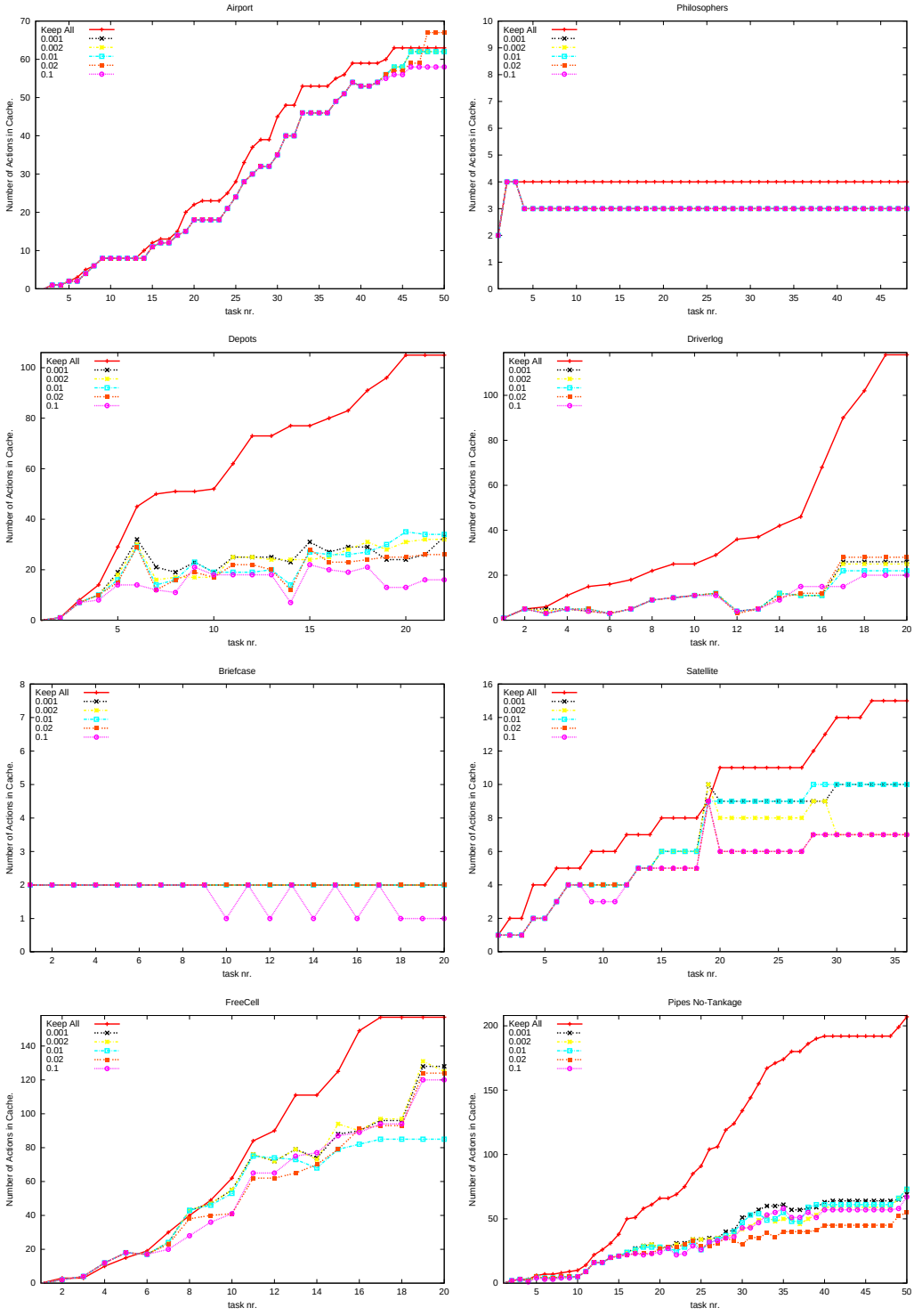


Figure 5.24: Number of macro-actions in the macro-action library using the instantiation versus use pruning strategy across a range of domains (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

equalled by two of the configurations of the time-out pruning strategy (see table 5.17). The common thing between the instantiation versus use pruning strategy and the successful versions of time-out pruning strategy that allows the planner to solve the problems in this domain more quickly is the smaller library size that the two maintain, whilst still keeping the macro-actions that are providing the best guidance on recent problems. The 0.01 version of the planner experiences difficulties in this domain as it learns a macro-action that causes it to enter a plateau with several dead end states that it cannot escape using only helpful actions. The 0.002 version shows improvement over the no macro-actions and keep all versions but does not offer the same improvement as the other versions because it takes a different path through search on one problem due to the pruning of an action and subsequent different trajectory through the search space. This different path taken in solving problem 19 is causes the planner to generate a larger macro-action library (see figure 5.24) which is not as helpful in solving problems. This is later pruned and becomes a similar size to that of the others on problem 29.

The most successful of all pruning strategies so far in the FreeCell domain is the 0.02 strategy, solving 19 of the 20 problems in the evaluation suite. Unfortunately, though, again this is due simply to the macro-action collection generated happening to guide EHC in a different direction that is sometimes nearer to the solution, rather than particularly by great design. There is no clear pattern indicated by the coverage and performance of any of the other versions suggesting that the peak value should be at 0.02. In general macro-actions in this domain lead the planner in a different direction in EHC and sometimes produce solutions to more problems; sometimes to fewer depending on the route that the macro-actions cause the planner to take. It is, however, pleasing to note that in a directed search space the macro-actions can sometimes make the performance better and sometimes worse; rather than always making the performance worse.

In the Pipes No-Tankage domain all versions solve more problems than the no macro-actions version and one configuration, 0.1, solves as many problems as the keep all version. None of the versions manage to surpass the coverage achieved by the keep all version, however. The macro-action library sizes for the pruning versions of the planner are, again, much smaller than that for the keep all version of the planner: the 0.1 version achieves the same coverage with a library only $\frac{1}{4}$ of the size. The time taken to solve mutually solveable problems is improved for all but one version of the planner, the 0.02 version; the worse performance of this version is mostly due to a single problem, problem 34, taking a long time to solve

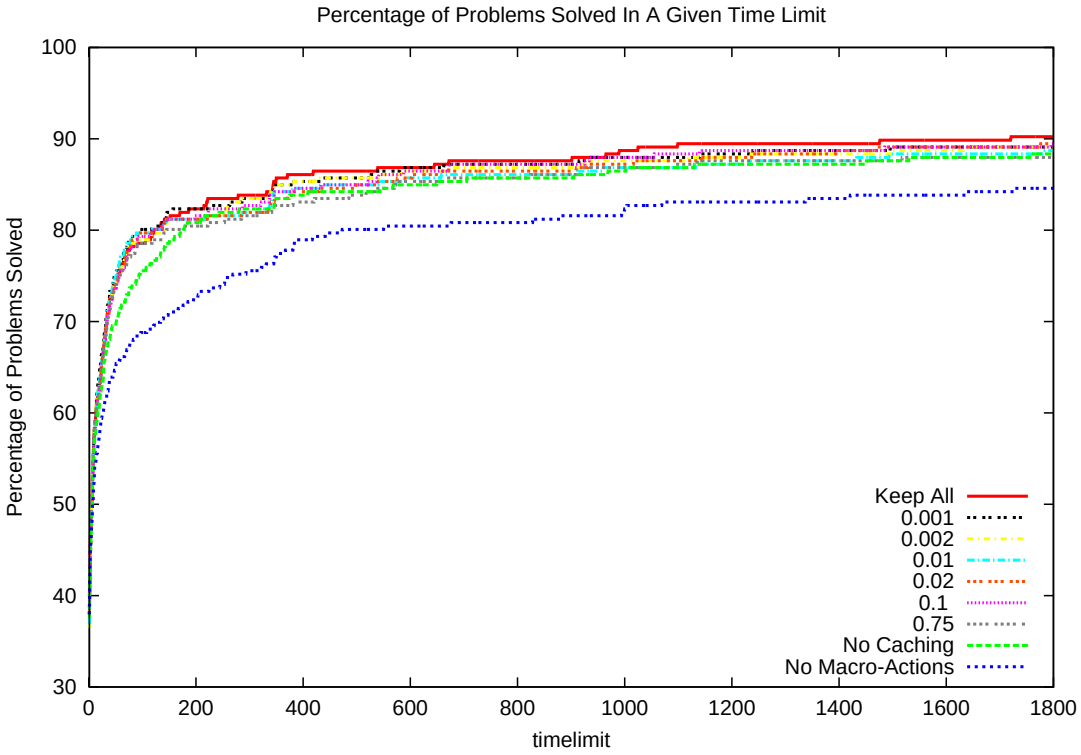


Figure 5.25: Percentage of Problems Solved in a Given Time Limit Using the Instantiation Versus Use Pruning Strategy

as a result of a macro-action leading the planner into a large plateau.

Conclusion

Instantiation versus use pruning can offer similar performance to keeping all macro-actions but with a much smaller macro-action library. In this instance keeping all macro-actions is able to be successful due to the aggressive search time pruning used by Marvin. However, if the macro-action libraries were to be used by a planner which did not prune so aggressively a smaller macro-action library that gives rise to similar benefits would clearly be preferable. In some domains, in particular those with directed search space, some configurations of the instantiation versus use pruning strategy are allowing the planner to solve more problems than were solved by all previous strategies. All versions of the instantiation versus use pruning strategy solve more problems than the no macro-actions control version. Overall this is a successful library pruning strategy generating small, high performing libraries; the benefits of the smaller size of these libraries are not, however, clear here because of the search time pruning employed.

Wilcoxon signed-rank tests do not show a significant difference between the versions presented in figure 5.23. If, however, instantiation versus use pruning is

made more aggressive, with higher use/instantiation ratios required for macro-actions to be kept, the performance degrades. This is because the macro-lists converge to containing few macro-actions other than those that have both use and instantiation counts of zero; leading the planner more towards the no macro-actions behaviour in many domains. A line showing the performance of a version using a cut off of 0.75 is shown in figure 5.25; the coverage achieved by this version degrades to below that of the no caching version by the end of the experiment. In the interest of being concise makespan data has not been included for these experiments; makespan improvements are similar to these observed in previous strategies.

5.3.6 Categorisation Based on Heuristic Profile

Plateaux fall into distinct categories: local minima, those in which all the neighbouring states have a heuristic value worse, or no better than the current state; and saddle points, those in which all neighbouring states have a worse heuristic value. The macro-actions represent the route off a plateau in a given situation; it is likely that when a similar situation arises again the heuristic profile observed will be similar.

Hypothesis

Considering only those macro-actions generated in similar heuristic situations will allow the planner to focus on only the best macro-actions in search. Specifically 'heuristic situations' are classified as either local minima or saddle points.

Analysis

Three sets of experiments have been done to test this hypothesis, the results of applying heuristic-profile based selection to different macro-action libraries. The experiments use different sizes of macro-action library: the first experiment, the results of which are shown in table 5.24 and figure 5.26, was done allowing the planner to cache all macro-actions generated whilst solving the problem. The second experiment is based on the survival of the fittest caching strategy keeping the top 10 macro-actions, results in table 5.24 and figure 5.27. The final experiment is done with no macro-action caching, results for this experiment are shown in figure 5.28. It would be expected that the performance of the versions of the planner keeping more macro-actions is affected more as they will have a larger library of macro-actions to be pruned. When only generating macro-actions on a

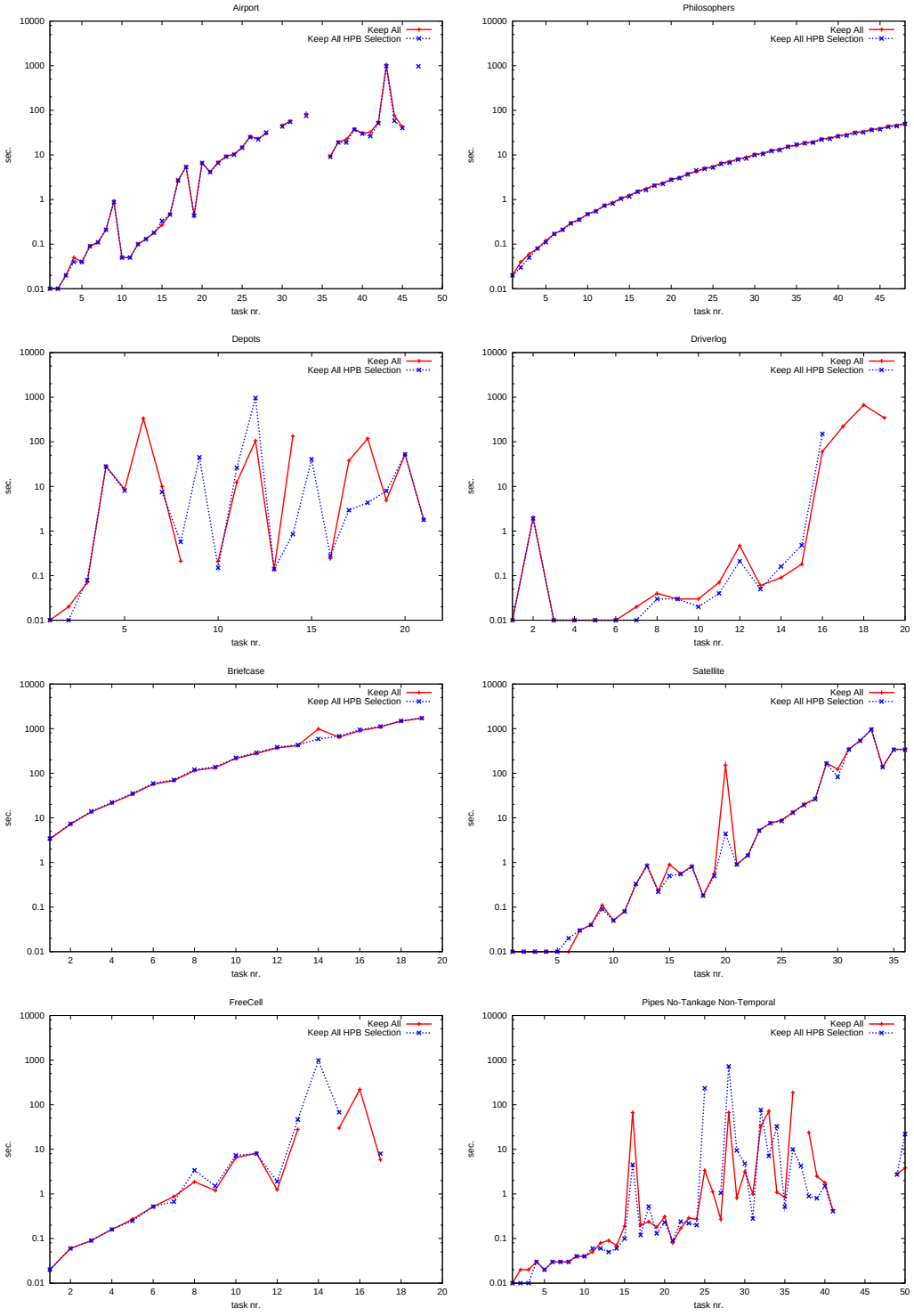


Figure 5.26: Time taken to solve problems in the evaluation domains using heuristic-profile-based selection, keeping all macro-actions (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	Keep All	keep All HPBS	Top 10	Top 10 HPBS	No Caching	No Caching HPBS
FreeCell	16	16	18	18	18	18
Airport	41	42	44	44	39	38
Depots	19	20	20	18	17	17
Philosophers	48	48	48	48	48	48
Driverlog	19	16	20	20	19	17
Pipes NT	42	42	42	39	39	39
Briefcase	19	19	19	19	19	19
Satellite	36	36	36	36	36	36
Totals	**240**	**239**	**247**	**241**	**235**	**232**

Table 5.24: Coverage Across the Evaluation Domains Using Heuristic Profile Based Selection (HPBS)

per-problem basis the planner will not have a large number of macro-actions to prune, so the pruning will not affect the behaviour of the planner as much.

Firstly, observing the results graphs, figures 5.26, 5.27 and 5.28, in conjunction with table 5.25 it is possible to observe that some domains exhibit no change in behaviour with and without the pruning due to the absence of different types of plateaux. In the Philosophers domain all of the plateaux are saddle points so no additional pruning will be done and the performance of the two versions is identical (slight differences in the no saddle point pruning version are due to noise). The Briefcase domain has the opposite property: here all plateaux are local minima, but again no additional pruning will happen and the performance is the same. In the Satellite domain we observe very similar performance as 99% of the plateaux encountered are local minima. There is, however, some difference in performance with the saddle point version noticeably avoiding the spike at problem 20 that was observed in the survival of the fittest, and other, caching strategies (see section 5.3.2). The slight variations in performance on other problems in this domain, for the keep all and top 10 versions, are due to the collection of macro-actions generated being different. The first difference occurs when not applying a saddle point action at a local minimum causes the planner to proceed in a different direction whilst doing search. Subsequently a slightly different collection of macro-actions is generated and the search on subsequent problems can potentially be made to proceed in different directions due a difference in which macro-actions are present. The libraries are, however, similar so most of the time search proceeds in the same way selecting the same macro-actions: those that are best in both libraries.

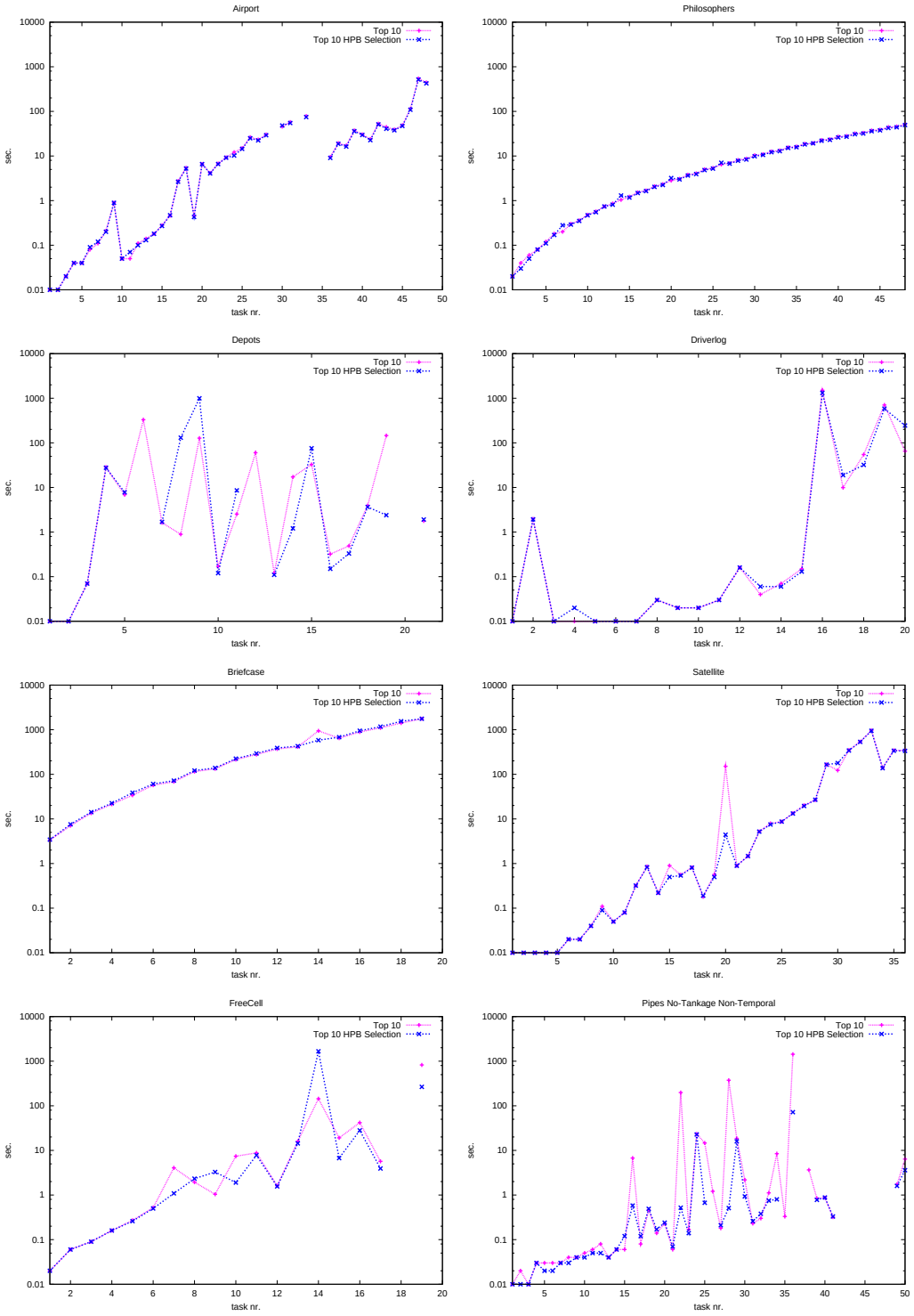


Figure 5.27: Time Taken To Solve Problems Keeping the Top 10 Macro-Actions, With Heuristic-Profile-Based Pruning(from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	% Minima Keep All	% Saddle Points Keep All	% Minima Top 10	% Saddle Points Top 10
FreeCell	52.57	47.42	50	50
airport	58.58	41.41	58.85	41.14
depots	66.66	33.33	77.38	22.61
philosophers	0	100	0	100
driverlog	58.49	41.50	49.80	50.19
Pipes NT	53.59	46.40	60.07	39.92
Briefcase	100	0	100	0
Satellite	99.04	0.95	99.04	0.95
Overall	**62.2**	**37.8**	**62.16**	**37.84**

Table 5.25: Percentage of Plateaux Encountered that are Saddle Points and Local Minima

Despite the 58%:41% split of plateau types in the Airport domain the performance of the two versions is rather similar. With a more even mix of each type of plateau one would expect the performance of the planner to be altered by this pruning strategy. A positive effect would be expected if macro-actions generated on saddle points were mostly useful at saddle points; a negative one if the heuristic profile at the point that the macro-action was generated is a poor predictor of when the macro-action is likely to be useful. Both the top 10 and keep all strategies have similar performance, however, to their respective counterparts using heuristic-profile-based selection. This can be explained by the fact that many of the problems in the Airport domain are solved by best-first search where the plateau-escaping macro-actions are not as effective as in EHC. Furthermore, all of the top 5 macro-actions are observed to be generated both at saddle points and at local minima, therefore most of the time the two versions will select one of the same few of macro-actions. The sixth ranked macro-action is used only 10 times, and subsequent macro-actions are used less. Most macro-action usages consist of the top 5 macro-actions: these macro-actions account for 63% of macro-action usages.

In the Depots domain, which has a roughly 2:3 ratio of local minima and saddle points the difference in performance is more marked. Figure 5.27 shows that the top 10 version of the planner using heuristic-profile-based selection solves most mutually solved problems faster than, or in the same time as, its counterpart not doing heuristic-profile-based pruning. The two versions each solve a problem that the other doesn't solve. When keeping all macro-actions, however, the heuristic-profile-based pruning causes the planner to solve one fewer problem: the problem

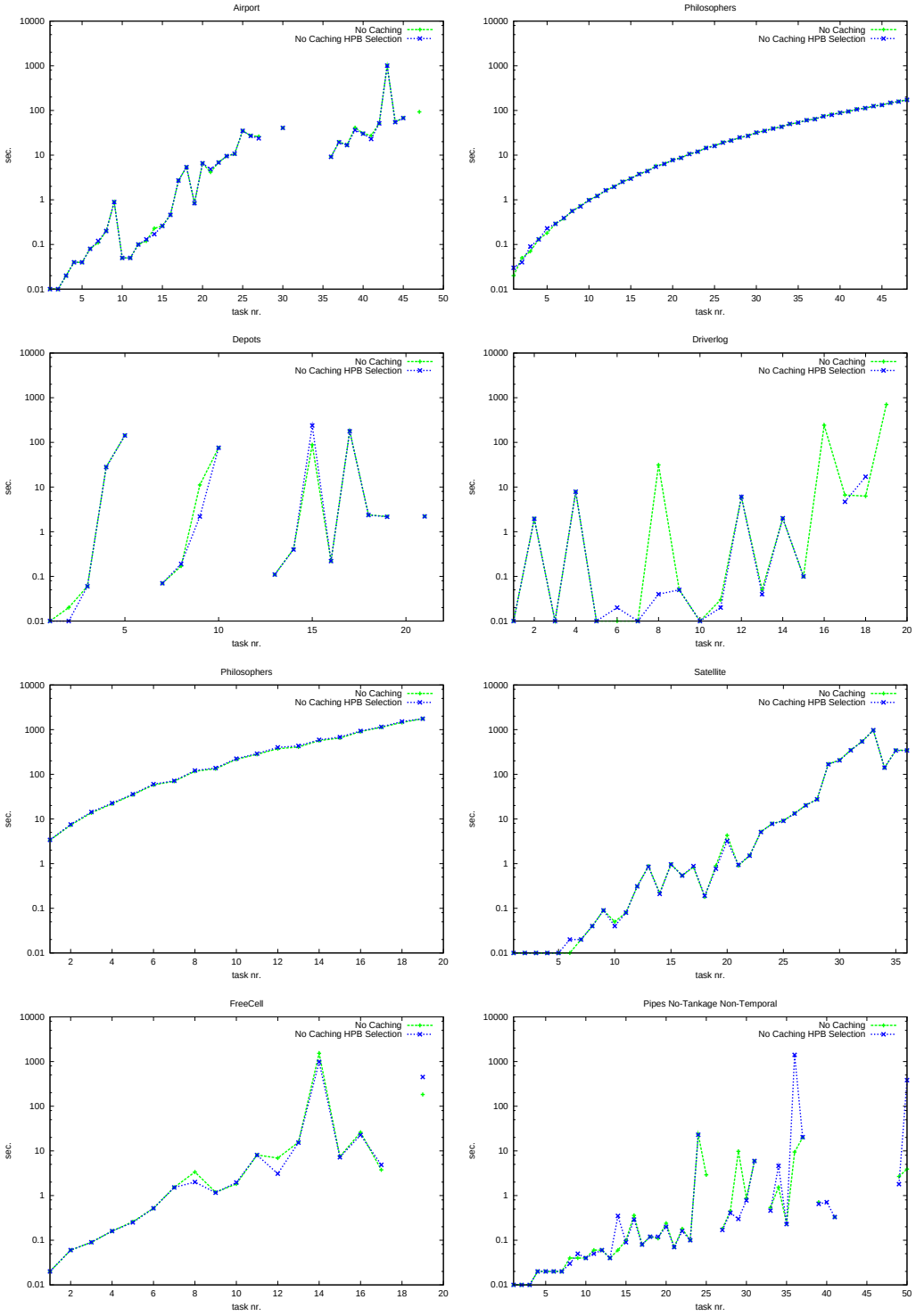


Figure 5.28: Time taken to solve problems in the evaluation domains using heuristic-profile-based selection and no macro-action caching (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

that is lost is problem 22. This is the same problem that is solved by the top 10 version using heuristic profile based selection and not by the top 10 version that does not use the strategy. The keep all version of the planner applies a saddle point macro-action at a local minimum and goes on to successfully solve the planning problem; the same version using heuristic profile based selection does not consider this macro-action and fails to solve the problem. There are other instances where the version of the planner prunes these macro-actions when they could have been used and is still successful: the failure is a combination of the natural direction taken by EHC (given the order in which the actions are given to the planner) and the macro-action library generated up to that point; as well as the caching strategy considered.

Decreased coverage is observed in the Driverlog domain when doing heuristic profile based selection in the keep all and no caching versions of the planner. The planner is pruning macro-actions in this domain at points in search where they are necessary to achieve the improved coverage obtained by these versions. There are a wide range of plateaux types in this domain and in order to gain maximum benefits in this domain macro-actions must be applied at points in search where the heuristic profile is different to that at which they were generated. The results in the FreeCell domain show similar performance between each strategy and its respective heuristic profile selection counterpart. Again the unpredictable behaviour observed using macro-actions in the FreeCell domain is seen with some problems being solved more quickly by each version than the other but no particular pattern emerging. This domain is another domain in which the split of plateau types is very even, but the profile information is not helping the planner to identify the best macro-actions.

In the Pipes No-Tankage domain the success of heuristic profile selection depends on the number of macro-actions being kept. In the keep all version the performance is more variable with the heuristic profile based pruning strategy causing some problems to be solved more quickly, by allowing the planner to focus on a smaller collection of macro-actions; and some problems more slowly, by pruning the macro-actions necessary to avoid resorting to best-first search. In the top 10 version, however, the heuristic profile based pruning is allowing the planner to solve most mutually solved problems more quickly. It is, however, the case that the coverage declines for the heuristic profile based pruning version to the same as that of the no macro-actions version. The effects that are being seen in this case, are in fact the effects of using, versus not using, macro-actions in this domain. The similarity of the behaviour of the planner to that of the no

macro-actions version, faster solutions but fewer problems solved, suggests that heuristic profile based selection is, in fact, pruning most of the macro-actions that would normally be used in search at the time they would be used.

Conclusion

Heuristic profile based selection of macro-actions has a slight negative impact on coverage and performance across the evaluation domains. The heuristic profile around the state from which the macro-action is generated is not a good indicator of whether it is likely to be reusable. Wilcoxon signed-rank tests do not show a significant difference between the heuristic profile based selection version and the corresponding version not doing heuristic profile based selection. Helpful macro-action pruning provides a very strong, and successful, suggestion of which macro-actions are the best to use in a given state. The guidance gained from considering heuristic profiles is much weaker and less reliable, pruning the actions in this manner before subjecting them to helpful macro-action pruning does not offer any benefits.

5.4 Generation and Use of Plateau-Escaping Macro-Actions Under Other Heuristics

This section contains discussion of the results obtained using plateau-escaping macro-actions under a different heuristic: Downward's causal graph heuristic [28, 29]. In producing results to test this hypothesis some domains other than the selected evaluation domains have had to be used. This is due to problems with the causal graph heuristic code crashing. The original code, taken from the planner Downward, was integrated into Marvin. Unfortunately the downward code cannot reason with two of the chosen domains as it crashes. The result is that when it is integrated into Marvin it cannot solve those problems making use of that code. This problem only affects two domains; these have been replaced, one with the same problems posed in a slightly different format, and the other with a domain that has similar properties; all of the other evaluation domains have been kept. The two domains in which issues arose are the Depots domain and the Briefcase domain. The Depots domain has been replaced with the untyped version and the Briefcase domain has been replaced with the gripper domain: a domain with very similar heuristic profile properties. The important issue is to maintain the spread of representative heuristic landscape profiles, rather than

specifically to keep exactly the same problems, this has been achieved through the selection of an appropriate replacement domain.

The survival of the fittest caching strategy is the strategy that has, so far, produced the best performing versions of the planner. This strategy has therefore been selected as the caching strategy under which to evaluate the performance using the CG heuristic[7].

The version of the planner used in this experiment is the default configuration of Marvin, except that the Causal Graph (CG) heuristic is used to guide search rather than the RPG heuristic. The preferred operators extracted from the CG heuristic are used instead of the helpful actions of the RPG heuristic; except in cases where no preferred operators are generated; when this happens the RPG helpful actions are used.

Hypothesis

The performance of a planner using EHC under a different heuristic can be enhanced through the use of plateau-escaping macro-actions. The performance gains seen will not necessarily occur in the same problems, or even in the same domains, due to the different heuristic profile and consequent trajectory thorough the search space, generated by the two heuristics.

Analysis

The first observation that can be made from the results in figure 5.30 is that, in general, the RPG heuristic is more consistent when using EHC than the CG heuristic: more problems are solved successfully by the no macro-actions version of the planner. The CG heuristic is generally more expensive to compute than the RPG heuristic. The Fast Downward and Diagonally Downward planners in IPC 4, that used the CG heuristic, were much more successful than standard FF is on those domains. It is clear, however, that much of the performance enhancement they gained was due to their overall architectures and search strategies. Diagonally Downward was able to benefit from the use of both the RPG and CG heuristics. Of course, the purpose of this section is not do do a detailed comparison of the two heuristics; this was done by Helmert [28] who showed that in best-first search the CG heuristic led to the expansion of fewer nodes in search.

[7] Although it would be interesting to see if the success of the caching strategies changes with the heuristic, repeating all the previous experiments here is not feasible.

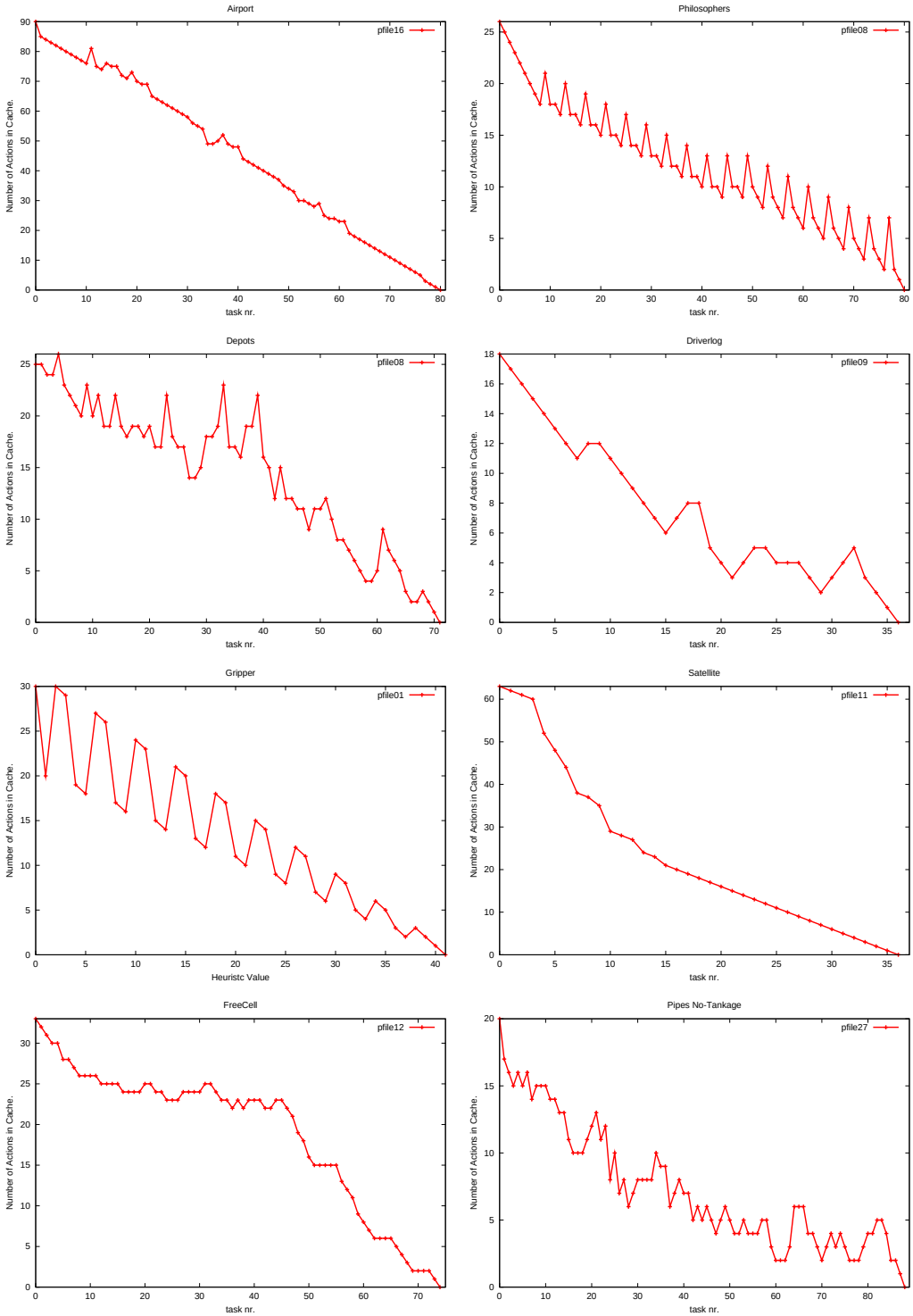


Figure 5.29: Heuristic Profiles of Evaluation Domains Under the Causal Graph Heuristic

Domain	Keep All	Top 10	Top 5	Top 2	No Caching	No Macro-Actions
FreeCell	12	15	17	17	17	16
Airport	22	22	24	24	22	22
Depots	15	16	17	20	17	11
Philosophers	48	48	48	48	48	42
Driverlog	18	20	20	16	19	17
Pipes NT	35	30	32	36	31	39
Gripper	20	20	20	20	20	20
Satellite	34	34	34	34	34	34
Totals	**204**	**205**	**212**	**215**	**208**	**201**

Table 5.26: Coverage Across Evaluation Domains Using The Survival of the Fittest Caching Strategy under the Causal Graph Heuristic

Airport

The version of the planner using the CG heuristic encounters fewer plateaux than that using the RPG heuristic in the Airport domain. The profile shown in figure 5.29 also shows that when plateaux are encountered they are generally slightly larger and require an upward step to be taken in heuristic value. It is the shorter plateaux that disappear under the CG heuristic, whilst the larger and more varied plateaux still remain present.

With larger plateaux one would perhaps expect the performance of the planner to be improved more dramatically by the use of plateaux escaping macro-actions. Recall, however, that the major difficulty in the Airport domain is, in fact, due to the dead end situations that arise, rather than the plateaux present. Indeed the standard RPG configuration using no macro-actions resorts to best-first search in order to solve 19 of the 38 problems it solves successfully (see section 5.3.2); the use of macro-actions in this domain was seen to improve the coverage achieved by the planner by allowing more problems to be solved by EHC rather than resorting to best-first search. Under the CG heuristic, however, the no macro-actions version of the planner resorts to best first search on all problems that it successfully solves. The other versions of the planner resort to best first search on all mutually solved problems; however, the top 5 and top 2 versions of the planner each allow two additional problems to be solved. Again, as when using the RPG heuristic, these problems are ones that are now solved by EHC, meaning that the planner does not resort to best-first search and subsequently fail to solve the problem.

Except for these two problems macro-actions are rarely successfully used in this domain under the CG heuristic. Figure 5.34 shows the mean use frequency of macro-actions of varying lengths. Interestingly a macro-action of length 66 is generated and subsequently used once. This use is, however, not an occasion on which the planner has realised that the macro-action leads to a better state and used it; simply an occasion where the plan has been post-processed and that same sequence of actions, representing a large plateau has been generated by best-first search. Macro-actions of length 2 are used, on average 4 times in this domain and macro-actions of other lengths are used only once on average. The macro-action library generated under the CG heuristic, the size of which is shown in figure 5.32, is shorter than that generated under the RPG heuristic. This is a result of the combination of two factors: firstly fewer problems are solved thus fewer plateaux are successfully escaped[8]; and secondly there are fewer plateaux encountered in this domain when searching under the CG heuristic.

Philosophers

The performance in the Philosophers domain under the CG heuristic mirrors that under the RPG heuristic. Figure 5.30 clearly shows a trend similar to that seen using the RPG heuristic: using macro-actions in this domain offers a consistent performance improvement; caching macro-actions offers a further performance improvement. The extra benefit seen under the CG heuristic is that extra coverage is achieved with respect to the no macro-actions version of the planner. This is, of course, because the version using no macro-actions is slower under the CG heuristic and so cannot solve all of the problems within the 30 minute time limit. The same effect would have been observed under the RPG heuristic had the time limit been reduced, or increasingly difficult problems been used.

The key difference between planning under the two different heuristics is the library of macro-actions generated. Recall that under the RPG heuristic three useful macro-actions are generated: one, of length 11 and one of length 3, each being applied for each pair of philosophers; and another of length 7 that is used once in problems with odd numbers of philosophers. The macro-actions correspond to the plateaux observed in figure 5.2. Under the Causal Graph Heuristic it is two four-step macro-actions that are the most used, as shown in figure 5.34. One of these is used n+1 times on the nth problem; and the other n-1 times for even numbered problems, and n-2 times for odd numbered problems. The two

[8]Macro-Actions are only stored when they occur in a successful solution plan to avoid learning bad action sequences.

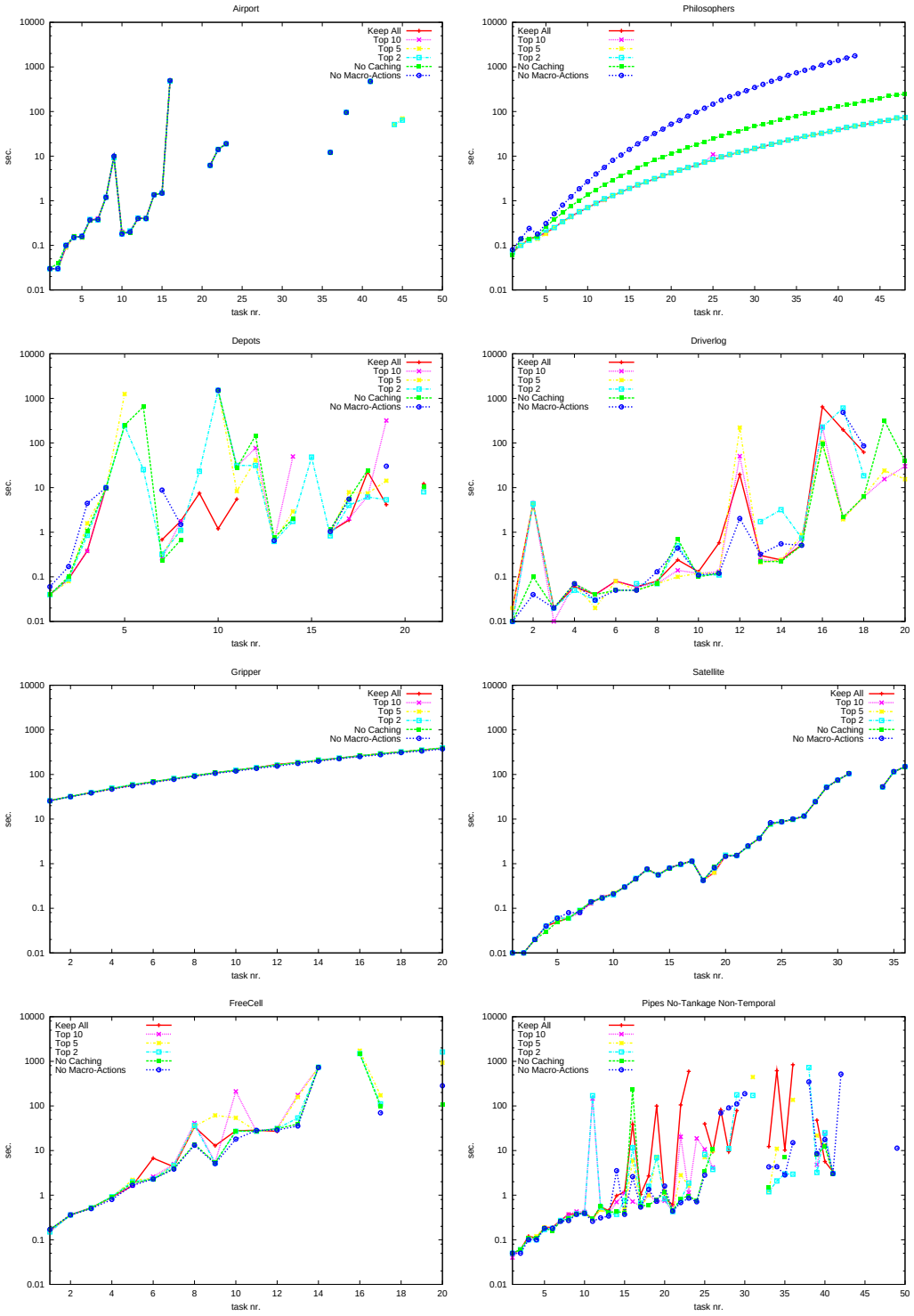


Figure 5.30: Time taken to solve problems in the evaluation domains using the survival of the fittest caching strategy under the CG heuristic

most common types of plateau can be seen in figure 5.29. There are also two macro-actions that are each used once on each problem instance. These are two plateaux that appear quite similar on the heuristic profile for the Philosophers domain in figure 5.29 but on close inspection it is possible to observe that the tenth plateau and the final plateau are different to all other plateaux encountered.

Although the plateaux under the RPG heuristic are different to those under the CG heuristic; both generate plateaux that are identical across all problems when the same heuristic is used. The CG heuristic, unlike the RPG heuristic, generates a macro-action that is applied to each individual philosopher. Most of the plateaux arising under this heuristic do not require the interleaving of activities involving two philosophers to escape. There is only one macro-action generated under the CG heuristic that does involve two Philosophers, this is the action that is applied once in the middle of the plan (corresponding to the tenth plateau in the figure 5.29 profile). As usual, in this domain, the makespans of plans produced by all versions (shown in figure 5.31) are identical and optimal.

Driverlog

The use of macro-actions shows benefits in the Driverlog domain under the CG heuristic. The no macro-actions version of the planner solves 17 of the 20 problems in the evaluation suite; whilst most of the macro-action versions of the planner solve more than this. The best performance is achieved by the top 5 and top 10 versions which successfully solve all 20 of the problems. The peak performance under the RPG heuristic was also achieved when caching the top 10 macro-actions.

Some of the macro-actions learnt are similar for both heuristics. For example the top macro-action generated under the RPG heuristic is unload-truck_drive-truck; whilst the CG heuristic generates the top macro-action drive-truck_unload-truck. These macro-actions do both, however capture a similar weakness in the heuristic: the fact that a package truck is often assumed to be in a location that it is not in, due to ignoring the delete effects. The unload-truck_drive-truck ordering forces the planner to move the truck after unloading a package to achieve the precondition that it is in the new location; whilst the drive-truck_unload-truck macro-action forces the planner to drive to another location before unloading a package.

Other macro-actions are quite different: the plateaux in this domain under the CG heuristic tend to be larger. Comparing figures 5.7 and 5.33 it is possible to observe that a number of larger macro-actions are generated under the CG

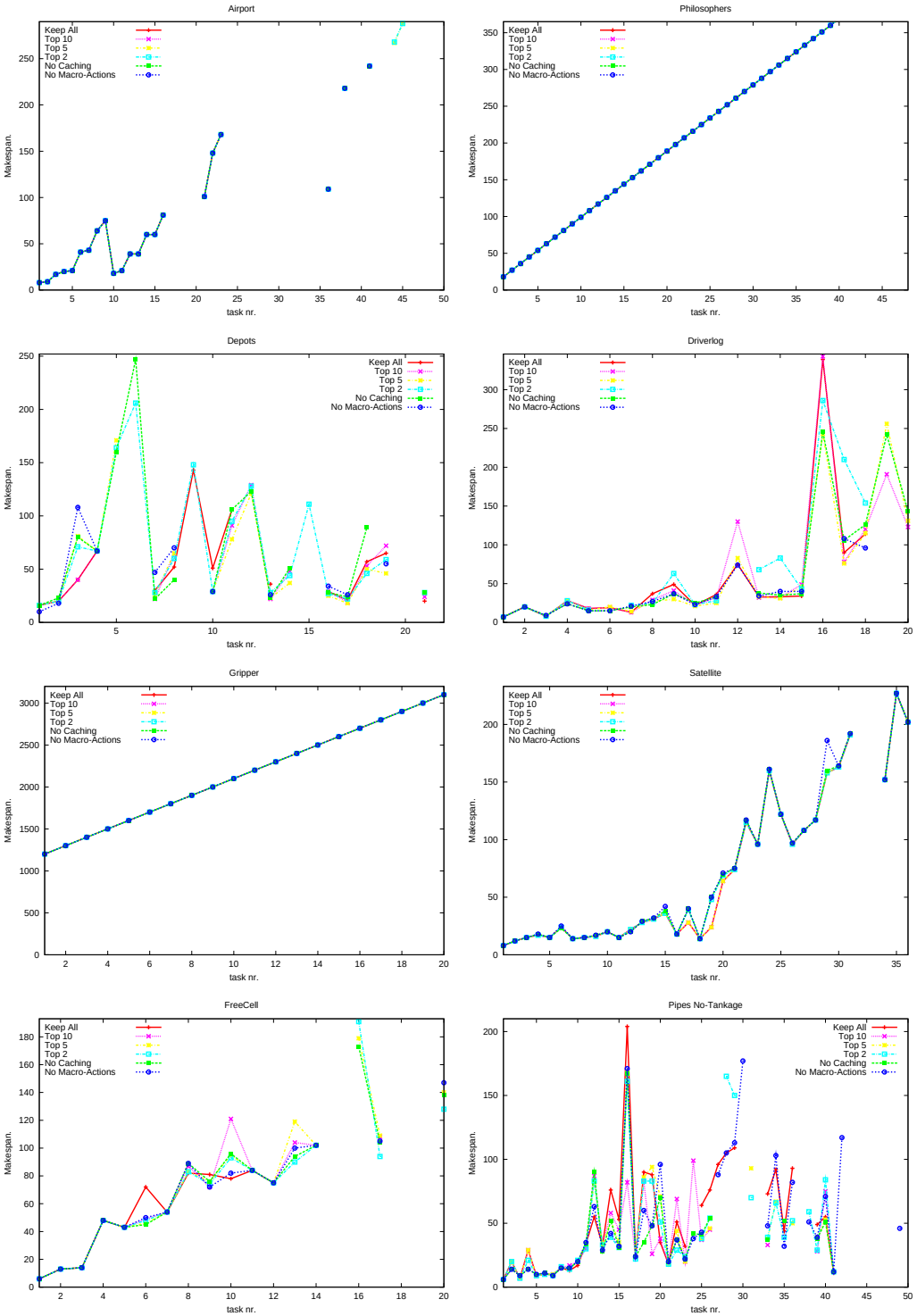


Figure 5.31: Makespan of plans generated caching macro-actions under the CG Heuristic

heuristic. Furthermore, observing figure 5.34, macro-actions of length 4 have the highest mean use count. In general the plateaux, and reusable action sequences, are longer under the CG heuristic allowing for greater time benefits to be seen when using plateau-escaping macro-actions. In the survival of the fittest strategy under the RPG heuristic the macro-actions did not give a time improvement on mutually solved problems in this domain; however here, with the larger plateaux on which macro-actions can be used, a time improvement is seen in all but the top 2 version. The time improvements are, in fact, reasonably large with versions of the planner reducing the time required to solve problems by between 19 and 36 seconds (see table 5.27).

The problems seen under the RPG heuristic with the macro-actions causing the planner to resort to best-first search are not seen here. In fact the macro-actions under the CG heuristic prevent the planner from resorting to best-first search: the no macro-actions version solves 10 of the 17 problems it solves successfully by best-first search; whilst the macro-action caching versions solve only 1 problem each by best first search.

The top 2 version of the planner actually performs worse than the no macro-actions version, both in terms of time taken and number of problems solved. The macro-actions it has learnt lead it into large plateaux from which it cannot escape. It is, however, the case that macro-actions are allowing the planner to do more search using EHC: all problems that are solved faster by the control than by the top 2 version of the planner are solved by the control using best-first search.

All version of the planner using macro-actions have slightly worse solution quality, with longer makespans on average, due to applying longer macro-actions to quickly escape plateaux on which a shorter exit path could have been otherwise found. The top 2 version, in particular, finds long plans as a result of bad guidance from macro-actions leading search into long plateaux.

Depots

The results in the Depots domain are generated using the untyped versions of the problems in the IPC 3 benchmark suite: these are the same problems as those in the RPG heuristic tests but with a different format[9]. The planner using the CG heuristic has not been as successful, in general, when solving the problems in this domain: the no macro-actions version of the planner solves only 11 of the 22 problems in the evaluation suite. The versions of the planner using macro-actions

[9]The reason for using the untyped version is purely pragmatic: the causal graph heuristic code crashes when using the typed version.

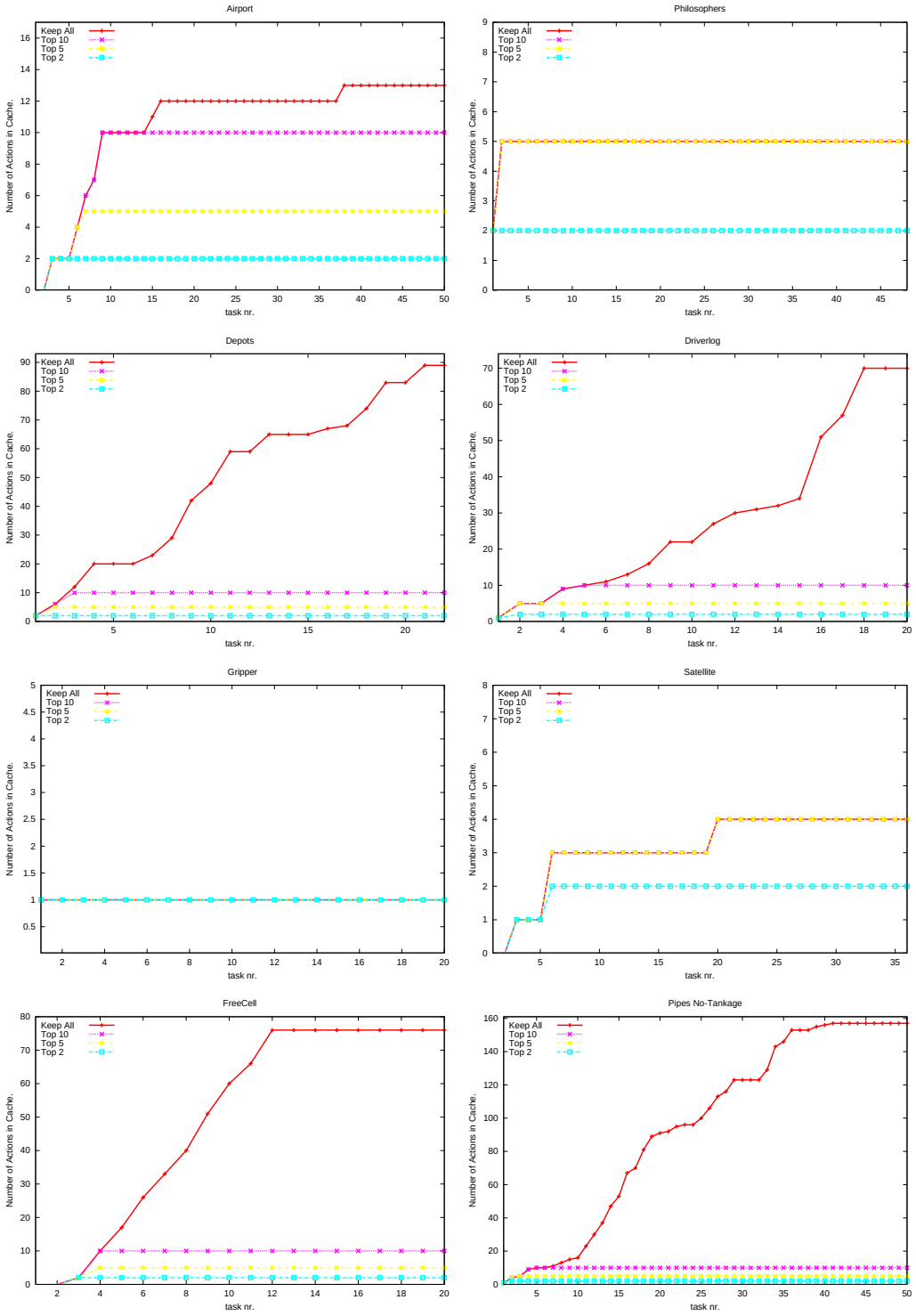


Figure 5.32: Number of Macro-Actions in the Macro-Action Library Generated Using the Survival of the Fittest Caching Strategy Under the Causal Graph Heuristic

Domain	**Keep All**	**Top 10**	**Top 5**	**Top 2**	**No Caching**
FreeCell	-3.46 *(12)*	-26.78 *(15)*	-62.19 *(16)*	-90.18 *(16)*	8.43 *(16)*
Airport	0.29 *(22)*	-0.01 *(22)*	0.54 *(22)*	0.56 *(22)*	-0.06 *(22)*
Depots	141.61 *(11)*	-25.15 *(11)*	1.90 *(11)*	3.5 *(11)*	1.01 *(10)*
Philosophers	315.41 *(42)*	315.32 *(42)*	315.4 *(42)*	315.28 *(42)*	291.07 *(42)*
Driverlog	17 *(17)*	30.00 *(17)*	19.93 *(17)*	-4.37 *(15)*	35.16 *(16)*
Pipes NT	-66.11 *(34)*	-6.42 *(29)*	-4.89 *(30)*	-15.9 *(34)*	-7.34 *(30)*
Gripper	-6.72 *(20)*	-7.38 *(20)*	-7.15 *(20)*	-7.31 *(20)*	-6.97 *(20)*
Satellite	0.19 *(34)*	0.19 *(34)*	0.11 *(34)*	0.18 *(34)*	0.18 *(34)*
Totals	**49.77** ***(192)***	**34.97** ***(190)***	**32.95** ***(192)***	**25.21** ***(194)***	**40.18** ***(190)***

Table 5.27: Mean of time taken by no macro-actions version minus time taken using each configuration of the survival of the fittest strategy under the CG heuristic. Results are calculated on mutually solved problems

do, however, give a great improvement in coverage allowing the planner to solve 20 problems in the best configuration, top 2.

Interestingly, under the CG heuristic the planner solves more problems in the versions keeping smaller numbers of macro-actions; whereas under the RPG heuristic it was versions of the planner keeping larger numbers of macro-actions that were most successful. Table 5.26 shows that the no caching and top 5 versions of the planner have equal coverage each solving 17 of the 22 problems; the peak coverage of 20 when caching the top 2 macro-actions and other configurations more distant from either side of this solve fewer problems. All macro-action caching versions do, however, successfully solve more problems than the no macro-actions version. The macro-action versions are allowing the planner to solve more problems without the need to resort to best-first search as in the RPG version.

The best macro-actions generated are similar under both heuristics. The top rated macro-action is the same: lift-load. The plateaux under both heuristics do actually represent similar action sequences. The appearance that they are dramatically different, when comparing figures 5.1 and 5.29, arises due to the fact

that the value generated by CG heuristic has a greater tendency to suddenly increase by a large amount when certain actions occur; whereas the value generated by the RPG heuristic is rarely observed to increase as steeply. The actual values that the heuristic takes on the plateaux are, however, not relevant in determining macro-action[10] the important factors are the length of the plateau and the final exit sequence found.

In terms of time taken to solve mutually solveable problems the keep all version of the planner is the fastest by a large margin. The top 10 version is actually slower than the control on mutually solveable problems; although of course if the other problems were allowed to complete and included the time difference would change very much in the opposite direction. The apparent worse performance of the top 10 strategy is mostly due to a spike in the time taken for one problem, problem 19. The makespan of the plans in this domain is improved by using macro-actions under the CG heuristic, as it was under the RPG heuristic (see figures 5.31 and 5.28).

Gripper

The Gripper domain was included in the evaluation for the CG heuristic instead of the Briefcase domain. This is due to difficulties in running the original CG code on Briefcase problems. The Gripper domain was chosen as a replacement because its heuristic profile is very similar to that of the Briefcase domain and the two domains have very similar properties. When searching in the Gripper domain the planner encounters many small plateaux and the same sequence of actions is required to escape each of them, as illustrated in figure 5.29.

The makespan of the plans generated, and indeed the plans themselves, are the same for all versions. They are, however, not optimal: the causal graph heuristic directs search in such a way that the planner does not take the optimal sequence of picking up two balls then moving then dropping both; it instead picks up and transports only one ball at once in all versions. The macro-actions do not improve this behaviour; nor do they make it worse. Only one macro-action is generated in this domain, this action is move-pickup-move. This is learnt because the planner selects this sequence of actions without using macro-actions; the suboptimality is not a result of the macro-actions but a result of the heuristic guidance. Since only one macro-action is generated, the performance of all of the caching versions of the planner is the same as all versions only have the same single macro-action in the library.

[10]Except that they will determine the path taken by best-first search.

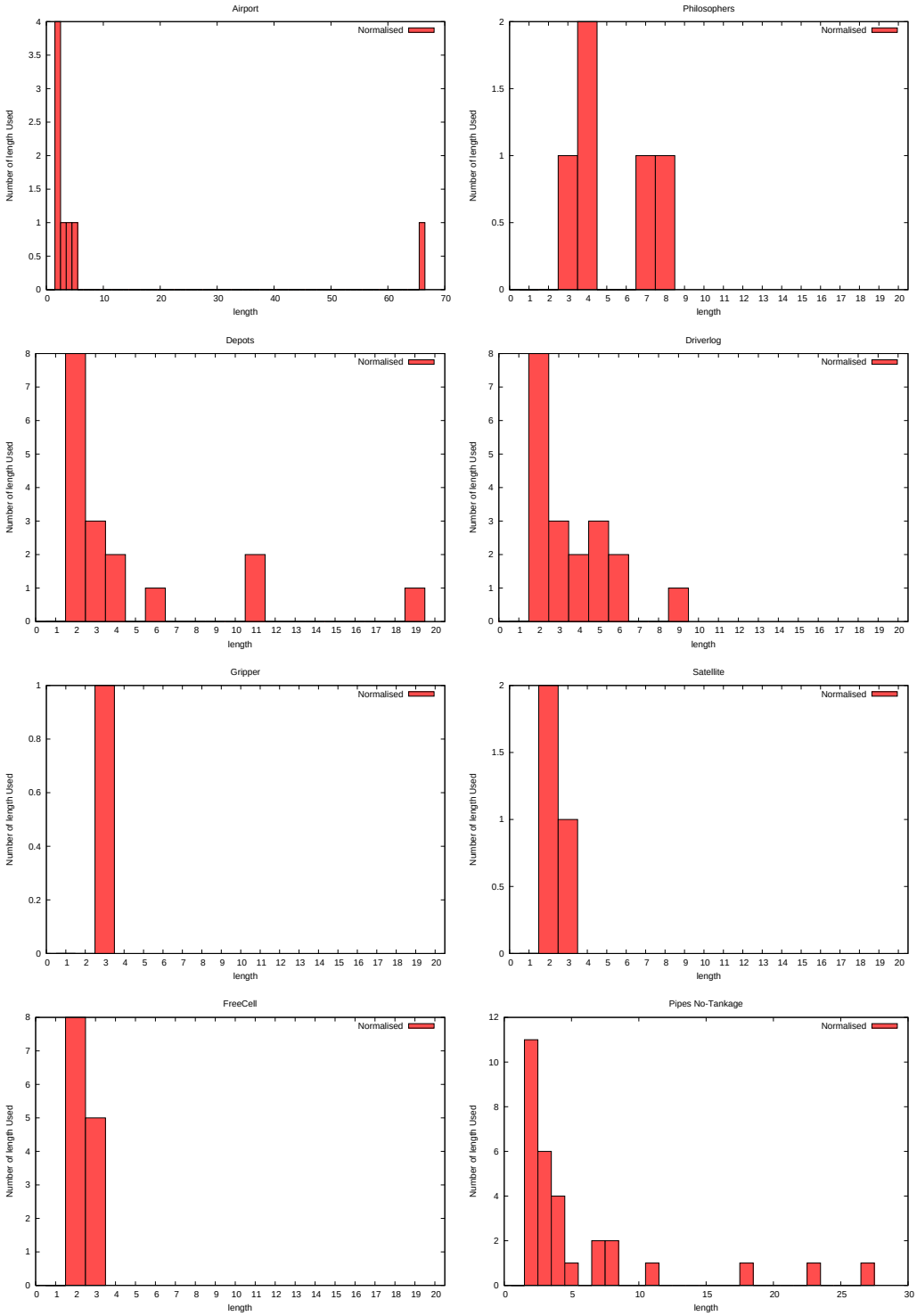


Figure 5.33: Number of Macro-Actions of Different Lengths Generated Under the CG Heuristic

Given the large number of similar plateaux in the Gripper domain the performance of the planner, shown in figures 5.30 and 5.27, is somewhat surprising. The versions of the planner using macro-actions actually solves problems slightly more slowly than the version using macro-actions. Macro-actions are, in fact, used to successfully escape every plateau encountered during search. The macro-action versions of the planner are expanding far fewer nodes in order to solve the problems: the keep all version of the planner is able to solve the final problem only expanding 3113 nodes; whereas the no macro-actions version expands 4662. The keep all version prunes 1558 states as a result of symmetric action pruning; the no macro-actions version prunes over 600,000 but these are never encountered by the macro-action versions of the planner. The increase in time taken is due to the increased load placed on the action applicability testing and symmetric action pruning mechanisms through the introduction of macro-actions.

Satellite

The results shown in table 5.27 for the Satellite domain show that planning with macro-actions offers a very slight time improvement over not planning with macro-actions. The macro-actions are successfully used to avoid exhaustive search on plateaux: the keep all version of the planner only encounters 4 plateaux in search during solving all the problems in the problems suite; whilst the version using no macro-actions encounters a total 169. The reason that no clear benefit is seen, despite exhaustive search being avoided, is that the planner is very successfully pruning symmetric action choices on plateaux, and the plateaux that are encountered are generally short (indeed figure 5.33 shows only plateaux of length 2 and 3 are encountered). The number of nodes expanded to escape from a plateau is generally not much, if at all, greater than the number of nodes expanded in other states. The macro-actions are, therefore, allowing the planner to avoid exhaustive search, but in this case not showing a great time saving because the exhaustive search being avoided is not expensive.

Figure 5.32 shows the length of the macro-action library generated. Only 4 macro-actions are generated and these are sufficient to avoid future exhaustive search on plateaux: recall that the keep all version of the planner only ever encounters 4 plateaux; these are the plateaux on which the macro-actions are generated. The most popular macro-action used here is, in fact, the same as the most used macro-action under the RPG heuristic: 'turn-to_take-image'. This macro-action is applied a total of 146 times; in fact it is the only frequently used macro-action: the remaining macro-actions are applied 2, 2 and 0 times.

Domain	Keep All	Top 10	Top 5	Top 2	No Caching
FreeCell	-1.65 *(12)*	-2.85 *(15)*	-1.42 *(16)*	2.19 *(16)*	0.19 *(16)*
Airport	0 *(22)*	0 *(22)*	0 *(22)*	0 *(22)*	0 *(22)*
Depots	5.73 *(11)*	8.18 *(11)*	6.73 *(11)*	5.73 *(11)*	8.3 *(10)*
Philosophers	0 *(42)*	0 *(42)*	0 *(42)*	0 *(42)*	0 *(42)*
Driverlog	-0.76 *(17)*	-3.22 *(17)*	1.35 *(17)*	-17.6 *(15)*	-1.31 *(16)*
Pipes NT	-5.38 *(34)*	0.48 *(29)*	0.03 *(30)*	-1.51 *(34)*	1.03 *(30)*
Gripper	0 *(20)*	0 *(20)*	0 *(20)*	0 *(20)*	0 *(20)*
Satellite	2.62 *(34)*	2.62 *(34)*	2.62 *(34)*	1.44 *(34)*	0.97 *(34)*
Totals	**0.08** ***(192)***	**0.74** ***(190)***	**1.33** ***(192)***	**-1.38** ***(194)***	**1.31** ***(190)***

Table 5.28: Makespan of Plan generated by No Macro-Actions Version Minus Makespan of Plan Generated by Each Version Using the Survival of The Fittest Strategy under the CG Heuristic

All versions solve only 34 of the 36 problems in this domain, which does not match the 100% coverage achieved by the RPG versions. The two problems that are not solved are a result of the planner resorting to best-first search almost immediately after search to solve these two problems commences. This occurs because the planner believes it is in a dead end due to having very few 'helpful actions' provided by the CG heuristic in the state reached.

A slight makespan improvement is observed in table 5.28 for each of the versions using macro-actions, particularly those caching macro-actions. The improvement is a result of small improvements across all problems with the plans generated by the macro-action versions being one action shorter on several problems. A large makespan improvement (visible in figure 5.31) is gained on problem 19 for the caching versions of the planner and on problem 29 for all the macro-action versions of the planner. This is a result of the macro-actions allowing the planner to escape a plateau with a shorter sequence than is generated using best-first search in the absence of macro-actions.

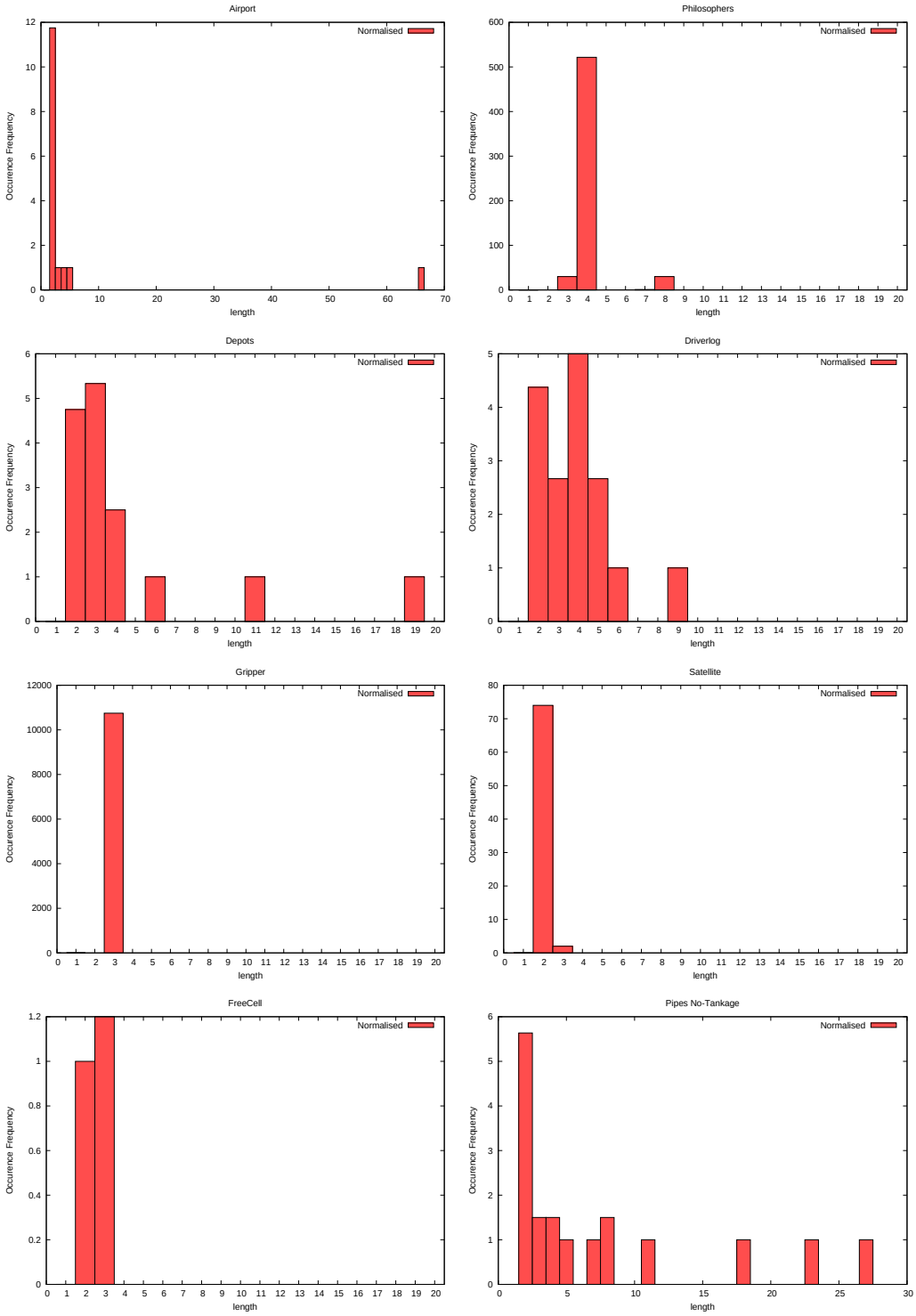


Figure 5.34: Frequency of use of Macro-Actions of Different Lengths Generated Under the Causal Graph Heuristic

FreeCell

The heuristic profile in the FreeCell domain is quite dramatically different between the two heuristics. Under the RPG heuristic the heuristic value suddenly increases dramatically after the application of a single action in many problems, as shown in figure 5.1. This happens much less frequently under the CG heuristic, the profile for the same problem under this heuristic is shown in figure 5.29. The rapid escalation in heuristic value, under the RPG heuristic, causes the planner to be on a plateau for a long time, gradually decreasing the heuristic value by the application of several actions. Interestingly if search had started from the second state the planner would not be on such a large plateau as the heuristic value generally decreases from this point.

In general, however, the RPG heuristic is giving a much higher, and more accurate measure, of the length of the solution plan from a given state in this domain. The RPG heuristic makes initial heuristic estimates that are, on average, 64% of the actual plan length; whereas the CG heuristic makes initial estimates that are only 43% of the plan length. Given the heuristic value rarely decreases by more than one step, per action that leads to a better state, this means that the CG heuristic version will reach more plateaux (or a similar number of larger plateaux). This suggests that the CG heuristic version should therefore perform worse in this domain. The coverage achieved by the CG version of the planner is, indeed, worse with the no macro-actions versions of the planner only solving 16 problems under this heuristic, two fewer than solved using the same configuration under the RPG heuristic.

The versions of the planner keeping smaller numbers of macro-actions solve one more problem than the no macro-actions version of the planner. In the two configurations keeping 10 or more macro-actions the coverage is reduced with only 15 problems solved by the top 10 version and only 12 by the keep all version. Under the RPG heuristic it was observed that macro-actions dramatically alter the path taken by EHC through the search space: sometimes the different path taken resulted in faster solution of the problem; and other times slower. Here, however, the path taken is more often a worse path.

Pipes No-Tankage

Under the RPG heuristic it was observed that macro-actions could improve coverage in this domain by allowing the planner to solve problems by EHC without needing to resort to best-first search. The planner resorts to best-first search fewer times under the CG heuristic: the no macro-actions version of the planner

	Top 10	**Top 5**	**Top 2**	**No Caching**	**No Macro-Actions**
Keep All (p =) Sig? Best	0.5364 No	0.7277 No	0.08253 No	0.0002963 Yes Keep All	0.08544 No
Top 10 (p =) Sig? Best		0.4603 No	0.0489 Yes Top 10	$8.211 * 10^{-05}$ Yes Top 10	0.04908 Yes Top 10
Top 5 (p =) Sig? Best			0.1839 No	0.000224 Yes Top 5	0.09932 No
Top 2 (p =) Sig? Best				$7.6 * 10^{-05}$ Yes Top 2	0.103 No
No Caching (p =) Sig? Best					0.00555 Yes No Caching

Table 5.29: Significance table using the survival of the fittest strategy under the causal graph heuristic: p is the probability that the null hypothesis cannot be rejected; sig? denotes whether or not the null hypothesis, that the versions take the same time to solve problems, can be rejected with probability > 0.95; Best is the best performing of the two configurations being compared

only resorts to best-first search 3 times and successfully solves 2 of these problems using best-first search. When using macro-actions the planner does resort to best-first search fewer times (never for top 5 and top 2; once for top10 and keep all; and twice for no caching); however, the macro-actions generated do not allow the planner to solve the problems via EHC within the time limit. The CG heuristic is a better guide for EHC in this domain: it does not lead the planner into dead ends as readily as the RPG heuristic. The macro-actions generated do, however, often lead the planner into large plateaux which are not escapable within the 30 minute time limit. The increased success of EHC and the fact that macro-actions can lead the planner into large plateaux, mean that the use of plateau-escaping macro-actions has worsened planner performance in this domain.

The macro-actions that are reused under the CG heuristic are much more varied in size than those under the RPG heuristic in the same domain. A macro-action of length 27 is successfully used in a solution plan. Under the CG heuristic

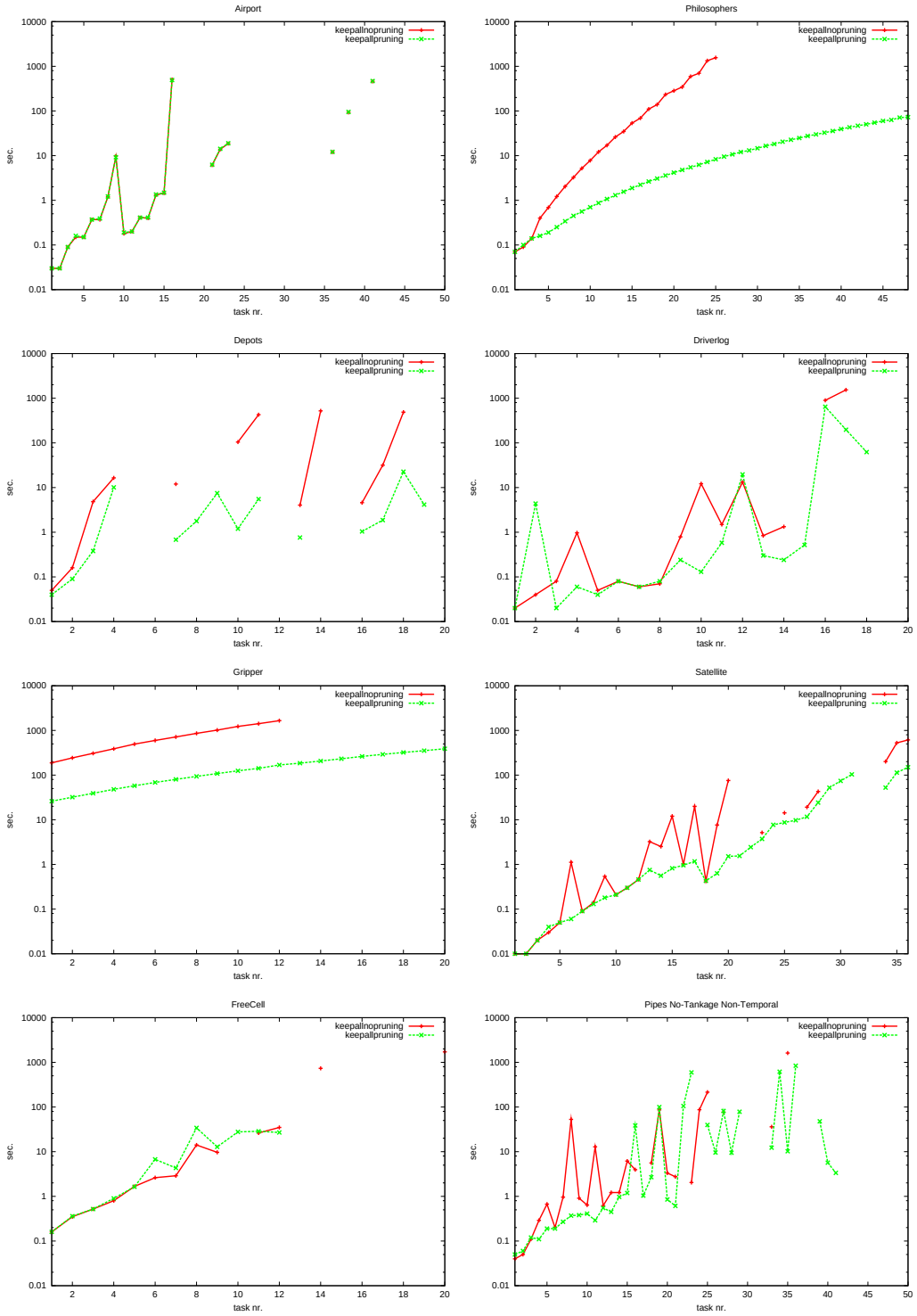


Figure 5.35: The Effects of Search Time Pruning Under the CG Heuristic

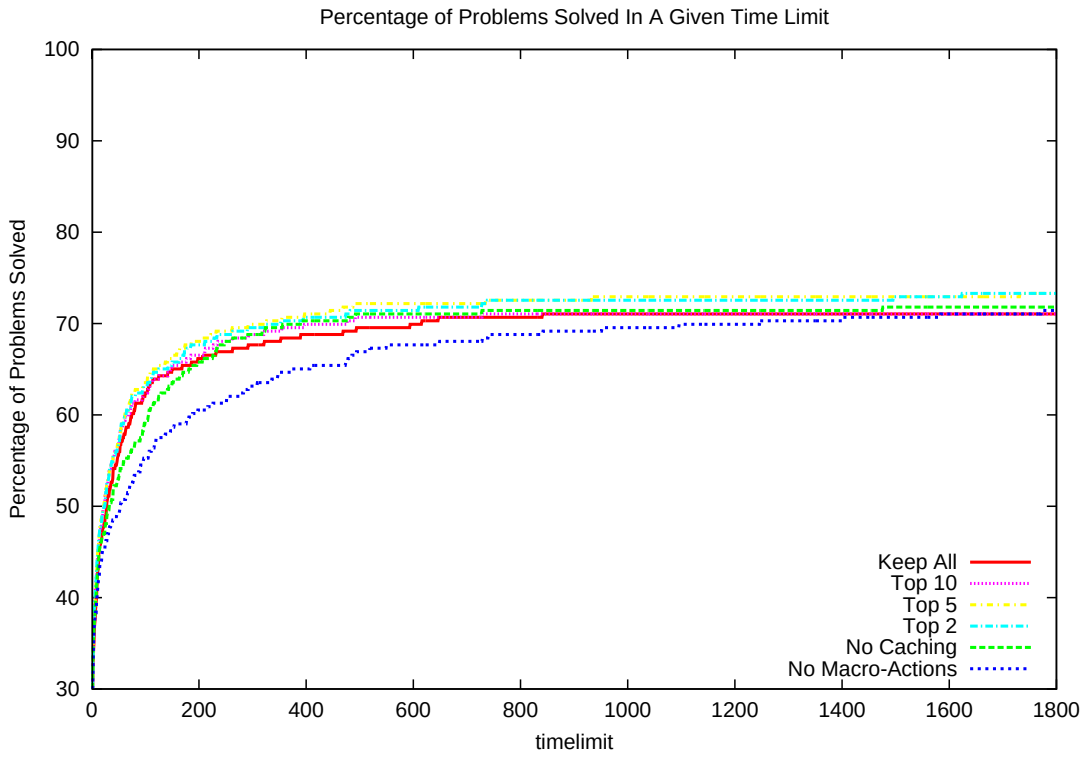


Figure 5.36: Percentage of Problems Solved in a Given Time Limit Using the Survival of the Fittest Caching Strategy Under the Causal Graph Heuristic

the most popular length of macro-action used in this domain is 2 with the macro-actions of greater length being used, on average, fewer than twice each. Under the RPG heuristic a macro-action of length 5 was generated and reused several times. This macro-action is not generated under the CG heuristic because the different relaxation does not give rise to the same plateau.

The makespan of plans generated using macro-actions in this domain show a slight improvement over the makespans of plans generated in the no macro-actions version for some configurations. Some versions, however, exhibit worse behaviour: the same macro-actions that cause decreased coverage by leading the planner onto large plateaux cause the makespan of some of the solutions to problems to increase as the planner is led in to large, but escapable, plateaux.

Conclusion

The use of macro-actions can improve coverage, and the speed with which plans can be found under the CG heuristic. When using the RPG heuristic it was observed that caching macro-actions could improve search time further over not caching macro-actions. This remains true under the CG heuristic, although the results are much more sensitive to the number of macro-actions cached. Figure

Length	Number of Times Used
2	338
3	10822
4	1065
5	10
6	3
7	3
8	33
9	1
11	3

Table 5.30: Usage data for macro-actions of length up to 15 Under the Causal Graph Heuristic. Missing data for a given length means no macro-actions of that length are generated.

5.36 shows that caching small numbers of macro-actions can allow performance to be improved. Caching large numbers of macro-actions does not allow the planner to perform as well as the versions caching smaller numbers of macro-actions; it does, however, still offer an overall improvement over using no macro-actions. This is a result that was expected when using the survival of the fittest strategy under the RPG heuristic, but was not observed due to the power of the search-time pruning. Figure 5.35 shows that under the CG heuristic, however, the search-time pruning is not as powerful so more improvement can be seen when macro-actions are pruned from the library. When using search-time pruning under the RPG heuristic, keeping all macro-actions, 44 extra problems were solved (see section 5.3.1); applying search time pruning under the CG heuristic allows only 35 more problems to be solved. The macro-action library pruning is therefore more important under the CG heuristic in ensuring that the planner is not presented with too many non-useful macro-actions.

Table 5.4 showing the results of statistical significance testing shows some surprising results: the top 10 version is shown to offer significant performance improvements over the other pruning versions of the planner; these results do not, however, take into account that the top 10 version has failed to solve several problems solved by the other versions. Similarly some versions cannot be shown to offer significant improvement, in terms of time taken to solve mutually solved problems, over the no macro-actions version; this is a result of the no macro-actions version failing to solve many of the problems solved by the other versions making it more difficult to show significance. Some versions of the planner do, however, show significant improvement over the no macro-actions version, the hypothesis that a library of macro-actions can improve performance therefore

holds.

The different relaxation used under the CG heuristic means that different macro-actions are generated and that macro-actions generated are of different lengths. The relaxation used by the CG heuristic ignores only some delete effects (not all as in the RPG heuristic) and the delete lists ignored can vary between problems and even states encountered in solving the same problem. Under the CG heuristic it is macro-actions of length 3, not 2, that are the most commonly used as shown in table 5.30. This further supports the hypothesis in section 5.2.3 that all macro-actions are interesting, not only those of length 2. The surge in popularity of macro-actions of other lengths observed under the CG heuristic is, however, mainly due to the use of macro-actions of length 3 and 4 in the Gripper and Philosophers domains respectively. These are both domains in which macro-actions are very useful and are used many times. When including these domains in the analysis 97% of macro-action use is of macro-actions of length greater than 2; when these domains are excluded, however, a different picture is seen with only 21% of macro-action use being represented by macro-actions consisting of more than 2 steps. This shows that macro-actions of length 2 remain more popular under the CG heuristic in the other domains, although they are by no means the only macro-actions of interest. Even when the Philosophers and Gripper domains are discounted under both heuristics the CG heuristic can be observed to use a more diverse range of macro-action lengths due to its more varied relaxation; whereas the RPG heuristic tends to favour the same macro-actions of length 2 more.

The makespan benefits observed under the RPG heuristic are not as clearly seen under the CG heuristic. All but one of the configurations show a makespan improvement; however this improvement is very slight, less than two time steps in all configurations. The one configuration that generates longer plans, top 2, generates plans that are only on average 1.38 steps longer. The magnitude of the impact of macro-action of makespan remains consistently small across almost all domains and configurations. The lessening of the positive impact of macro-actions on makespan is a result of the weaker helpful macro-action pruning. The planner is not guided as well to find the appropriate macro-actions to use. This phenomenon was also observed in section 5.3.1 when disabling search time pruning under the RPG heuristic, where the helpful macro-action pruning version of the planner generated plans, on average, 11.85 time steps shorter.

5.5 Simulating the use of Macro-Actions Through Action Reordering

The results obtained using the different action reordering techniques are discussed in this section. The strategies were described in detail in section 4.4. All of the action reordering strategies alter the order in which states are visited by Marvin during planning. The most basic strategy simply reorders the list of helpful actions that are being considered for application during search. Later strategies add best-first search on plateaux based on probabilistic data. All results shown in this section are produced using the sequential version of the planner so that it is clear which action precedes a given action and thus the appropriate data to update. The macro-action generation and use features of Marvin are disabled in this series of experiments so that the benefits of action reordering can be tested independently.

5.5.1 Basic Action Reordering

The basic action reordering strategy dictates the order in which actions are considered for application during search, these update strategies are described in detail in section 4.4.2. The data are updated according to one of three different strategies, when adding an action to the end of the plan update the probability that it follows:

- U1: The action that was previously at the end of the plan;
- U2: The action that was previously at the end of the plan if, and only if, the two actions share a common parameter;
- U3: The action in the plan that shares a parameter with the newly added action and all other actions following it in the plan can be mutually parallelised, if such an action exists. That is, an action that is in the collection of actions at the end of the plan that are ordered in such a way that is purely arbitrary since they are independent of one another.

The tests have been carried out in two configurations: the first using only action reordering data generated on the problem instance being solved; the second caching the probabilistic following data between problems.

5.5.2 Reordering Based on Information from the Current Problem Instance

This section presents the results of running the planner using action reordering data generated on each problem instance. The action reordering data is simply used to change the order in which the actions are considered for application during EHC search. The reordering is applied at all points during search. This reordering is based on experience of which actions have followed other actions so far in the plan; it will allow the benefits of simulating two-step macro-actions without potentially adversely affecting the performance of the planner through the addition of extra actions.

Hypothesis

Through reordering actions to consider the actions in an appropriate order for EHC search performance gains can made.

Analysis

The results shown in figure 5.37 show the time taken to solve problems using reordering with each of the three update strategies and the time taken by the control. The control is the sequential version of the planner using no macro-actions and no action reordering strategy.

In the Airport domain the planner manages to solve three more problems using the action reordering strategies: the control version of the planner only solves 38 of the 50 problems; whereas each of the reordering strategies allows the planner to solve 41 problems (see table 5.31).

The additional problems solved using the reordering strategies are problems 28, 31 and 33. In these problems the action reordering strategies allow the planner to avoid getting into deadlock situations, those in which two planes mutually block each others' forward progress due to the exclusion zones around them. Planes cannot reverse thus, if this situation occurs the planner must abort EHC and resort to best-first search. The action reordering strategies cause the planner to consider actions in a different order which does not lead to this situation in these three problems. The reordering strategies therefore allow the planner to successfully solve these problems using EHC; using best first search to solve the problems is not successful within the 30 minute time limit.

The data for mutually solved problems shows that the reordering configurations are on average between 1 and 2 seconds slower at solving problems; however,

Domain	control	U1	U2	U3
FreeCell	18	18	18	18
Airport	38	41	41	41
Depots	15	15	15	15
Philosophers	48	48	48	48
Driverlog	17	17	17	17
Pipes NT	37	37	37	38
Briefcase	17	17	17	17
Satellite	36	36	36	36
Totals	**226**	**229**	**229**	**230**

Table 5.31: Coverage Across Evaluation Domains Using Action Reordering with Different Update Strategies

the three reordering configurations have successfully solved three problems in under 40 seconds that the other configuration was unable to solve in 30 minutes. These problems were not included in the average as the control did not successfully solve them; nonetheless it is clear that had the control been allowed to complete and the time taken been included the reordering configurations would have been shown to have much improved performance. The different reordering strategies perform similarly to each other in this domain, except that the U3 strategy manages to solve problem 25 without resorting to best-first search. As this is a relatively small problem instance all of the other configurations successfully solve the problem after resorting to best-first search, however this process does take slightly longer.

In the Philosophers domain the performance of the planner can be seen to be consistently improved over the problems in the evaluation suite. Solving the problems up to that with 51 philosophers gives the versions doing action reordering a mean improvement in time taken of just over 12 seconds. It can be seen from the graph in figure 5.37 that the line representing the time for each of the configurations to solve problems is growing according to a regular function and that extending the problems to consider increased number of philosophers would result in greater and greater improvements for the versions using reordering strategies.

Planning in the Philosophers domain is very structured and the same sequences of actions occur many times. The planner is able to benefit from using action reordering data in the segments of search where the planner can find a strictly better state quickly: for example, at the start of the plan an 'activate-trans' action is required for each philosopher in the problem. This corresponds to a period of search finding a strictly better state at every choice point and

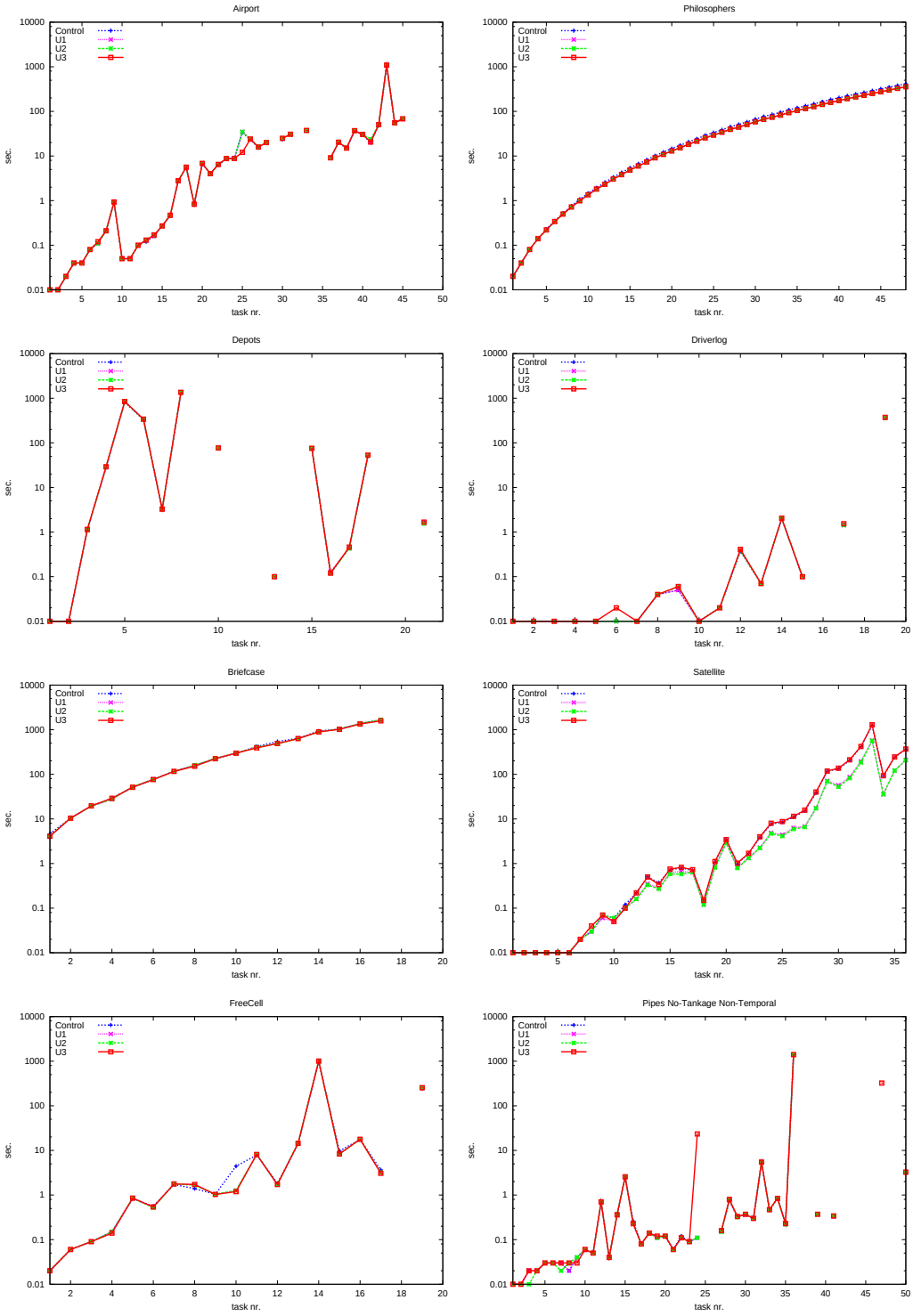


Figure 5.37: Time taken to solve problems in the evaluation domains using the different action reordering update strategies (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

is the steep descent section at the beginning of the heuristic profile shown in figure 5.2. The planner does not make the same gains that are made by using macro-actions as the action reordering data does not offer a major benefit during best-first search on plateaux. The search will have to explore any possibilities with equal heuristic value before others regardless of whether the action ordering data suggests that it is a good path to take through the search space.

In the Depots and Driverlog domains the action reordering strategies have very little impact on performance. Although it appears in table 5.32 that the version using no reordering performs better than the version using reordering in Driverlog, the difference in performance is only on one problem, problem 19. The only reason for the improved performance in problem 19 is that EHC fails using both configurations and the planner resorts to best-first search. The version using no action reordering is not able to make as much progress using EHC and thus fails slightly more quickly resorting to best-first search sooner. A similar phenomenon occurs in many problems in the Depots domain.

In the Briefcase domain a slight improvement in performance can be observed on some problems, however, the domain only consists of three actions, the ordering of which is frequently interchanged so although reordering has some positive impact it does not greatly enhance performance. The planner is unable to solve further problems after problem 17 as the memory requirements exceed the imposed limit.

In the Satellite domain the U1 and U2 versions of the planner demonstrate a significant performance improvement over the control version. EHC search in this domain consists mostly of states where a strictly-better state can be found at each choice point with a small number of plateaux that are escapable in very few (usually two) action steps. With good action ordering guidance the planner is able to proceed more quickly though this search giving a performance improvement.

The version using the U3 strategy is however learning closely the mistakes that the planner makes in this domain when guided by the RPG heuristic. Quite often, despite the fact that a Satellite can turn directly from any phenomenon to any other phenomenon, a chain of 'turn-to' actions are inserted into the plan. The U3 reordering strategy learns from this, since it looks in detail at previous actions in the plan to find a predecessor action, and suggests in the future using a 'turn-to' action to follow another 'turn-to' action. The other learning strategies pick up this phenomenon less frequently as a different action, involving a different entity, is often applied between 'turn-to' actions in a chain. Although the performance of the planner is not greatly improved by using the U3 reordering strategy it is not

Domain	U1	U2	U3
FreeCell	-2.36 *(18)*	-2.54 *(18)*	-2.08 *(18)*
Airport	-1.46 *(38)*	-1.66 *(38)*	-0.85 *(38)*
Depots	-5.53 *(15)*	-5.4 *(15)*	-6.01 *(15)*
Philosophers	12.42 *(48)*	12.21 *(48)*	12.12 *(48)*
Driverlog	-0.71 *(17)*	-1.03 *(17)*	-0.85 *(17)*
Pipes NT	-1.21 *(37)*	-1.05 *(37)*	-1.74 *(37)*
Briefcase	5.25 *(17)*	5.96 *(17)*	14.94 *(17)*
Satellite	41.83 *(36)*	43.01 *(36)*	-1.90 *(36)*
Totals	**6.02** ***(226)***	**6.18** ***(226)***	**1.7** ***(226)***

Table 5.32: Mean of time taken by control version minus time taken using each update strategy with action reordering. Results are calculated on mutually solved problems

made significantly worse: the planner is only learning from the mistakes it would usually make, the mistakes are still made in search whether they are reinforced or not.

In the FreeCell domain the performance of the planner is not greatly improved or worsened by the reordering strategies. Problem 10 is solved slightly faster by all of the configurations using reordering techniques; whereas problem 14 is solved slightly more slowly by these configurations. This is due to the action ordering the planner is using being better suited to those particular problems at that time, much like the chance direction taken using macro-actions. All other problems are solved in the same length of time for all configurations. In the Pipes No-Tankage domain the performance of all the configurations is again similar. The U3 version of the planner succeeds in solving one more problem than the other versions, problem 47. All other versions reach a plateau that is not escapable within the 30 minute time limit; due to the different order in which actions are considered the search the U3 version proceeds in a different direction and does not encounter the same plateau.

Conclusion

In some domains the action reordering strategy can offer a slight improvement in performance, in others the results are more significant. The action reordering data is not making a great deal of difference to performance in some domains as the data is only being learnt during planning on one problem instance so is not sufficiently strong to guide the planner. Furthermore the impact of the strategy is difficult to see when search resorts to exhaustive best-first search. It is pleasing to note, however, that the action reordering data very rarely has a negative impact on planner performance; with performance impact being positive in most cases. Statistical tests to demonstrate the correctness of the hypothesis will be presented in the following section, along with those for caching action reordering data, to allow a total ordering to be placed on the success of the strategies.

In these experiments the different probabilistic update strategies all exhibit very similar performance except in the Satellite domain. The action reordering techniques have very little impact on the makespan of solution plans with plans of similar length being generated by all configurations. The makespan of plans generated by the versions using the reordering strategies are, on average, approximately 0.5 timesteps shorter overall, this does not represent a significant change.

5.5.3 Caching Reordering Data

In this section the configurations of the planner presented in the graphs (in figure 5.38) differ from the previous section only in that the action probability data is cached between problems. The caching allows the planner to learn the appropriate ordering of actions over a number of problems. Further the planner can learn from successful plans that have completed in the past rather than from experience in solving the current problem which may, of course, not represent the correct way to reach the goal.

Hypothesis

Caching action reordering data can further enhance the performance of the action reordering strategy by allowing the planner to learn the best action to follow a given action over a larger number of problems.

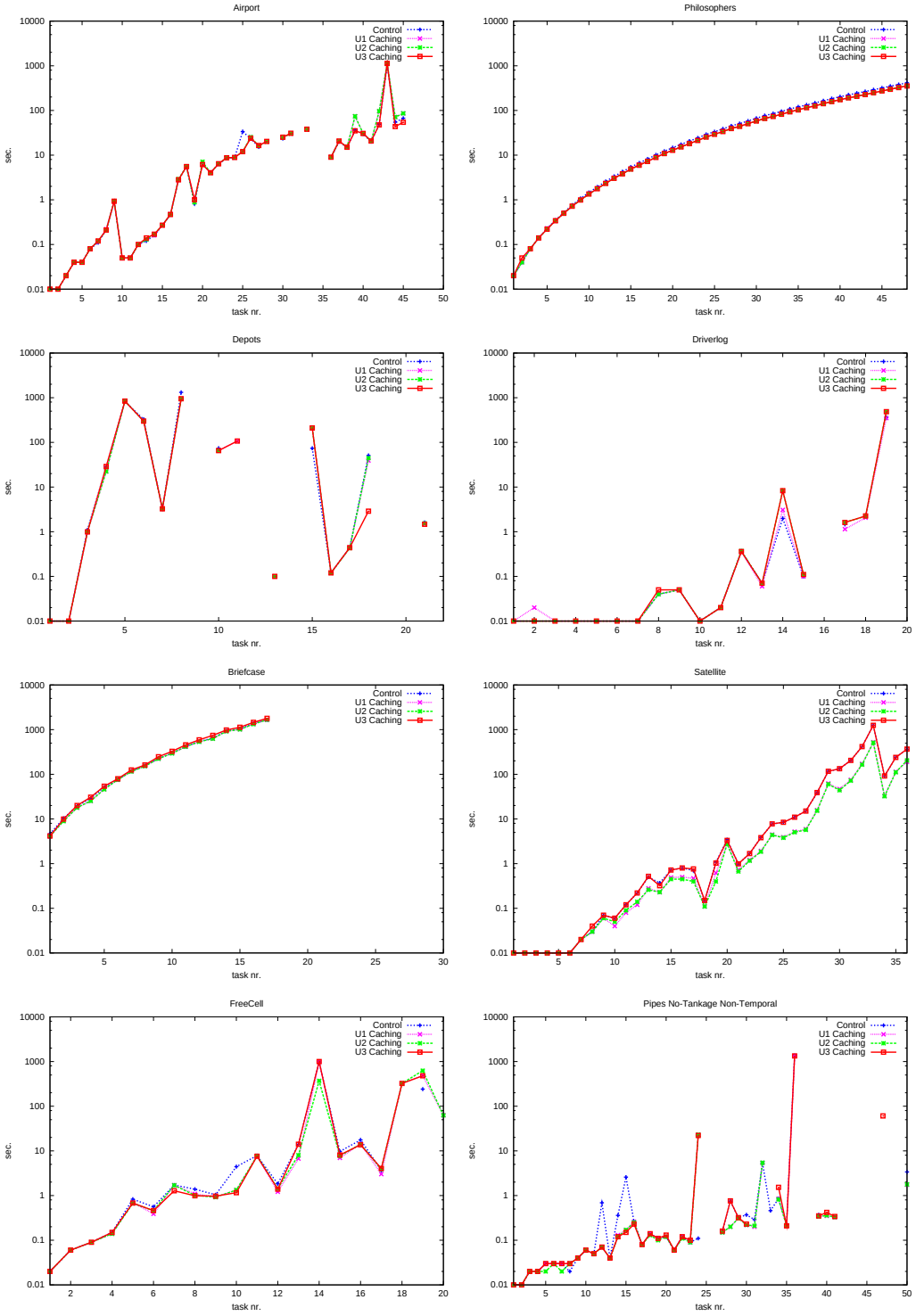


Figure 5.38: Time taken to solve problems in the evaluation domains using the different action reordering strategies caching probabilities between problems (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	Control	U1 Caching	U2 Caching	U3 Caching
FreeCell	18	20	20	19
Airport	38	41	41	41
Depots	15	16	15	16
Philosophers	48	48	48	48
Driverlog	17	18	18	18
Pipes NT	37	37	36	35
Briefcase	17	17	17	17
Satellite	36	36	36	36
Totals	**226**	**233**	**231**	**230**

Table 5.33: Coverage Across Evaluation Domains Using Action Reordering with each of the Update Strategies, Caching Reordering Data

Analysis

Overall the results for this configuration are more positive: caching the action reordering data is allowing the planner to learn from previous experience. Since the previous learnt information is generated from successful problem solutions it represents successful decisions from which to learn. Furthermore, the information is presented to the planner at the start of search and can be used throughout solving the problem; in the case where the planner learns based on each problem instance it must first make a reasonable advance in solving the problem before the probabilistic data becomes meaningful.

Table 5.33 shows that the planner solves additional problems in 5 out of the 8 evaluation domains when caching reordering data. In two of the remaining three domains, Philosophers and Satellite, all problems are solved but a performance improvement is seen for at least two of the configurations. This indicates that should further more difficult problems be posed the reordering versions are more scalable. In the remaining domain, Briefcase, all configurations of the planner are not able to solve the final three problems as a result of exceeding the memory limit imposed on the search.

The performance in the Airport domain is very similar to that exhibited by the equivalent versions not caching action data. The same number of problems are solved because the action data discovered in each problem is sufficient to guide the planner away from making the decision that causes deadlock of planes and forces the control version of the planner to resort to best-first search. When solving problem 39 the U3 caching version is able to maintain the performance that was exhibited in the no-caching versions and by the control. All three versions resort to best-first search within a second of each other, however the fact that the actions

are considered in the suggested order in best first search means that the U3 and control versions reach the solution in the search space more quickly.

In the Briefcase domain the technique is again showing a slight positive impact in the U1 and U2 versions with a negative impact being observed in the U3 version. In the U3 version the performance is degraded as the planner is treating the subgoals with too much independence due to the update strategy. The U1 and U2 versions learn that when putting an object in a briefcase the next thing to do is to put another object in the briefcase. The U3 version, however, learns to consider moving the briefcase next, which will leave uncollected items at locations which must then be collected.

All versions of the planner have solved all the problems in the Philosophers and Satellite domains therefore the data in figures 5.32 and 5.34 are directly comparable (the control used is identical). The performance in the Philosophers domain is almost identical to the performance of the no caching versions. Search in this domain follows a regular pattern of action ordering that the planner is able to learn quickly on each problem. The caching versions have a very slight performance improvement brought about by the search guidance being available from the very start of problem solving. The Satellite domain shows a similar pattern but with a greater performance improvement being gained by caching the data, as opposed to learning it on each problem. The version using the U3 reordering has improved by caching action ordering to give a slight performance enhancement; rather than the slight degradation that was found without caching. Caching the data is allowing the planner to learn from good decisions on smaller problems, in which less redundancy is introduced. Caching the helpful data in the U1 and U2 strategies improves the mean performance of the planner by around 4 seconds. In these two domains the makespans of the plans generated are almost identical, slightly improved in Satellite and identical in the Philosophers domain, so the planner is finding plans of the same quality in less time.

In the Driverlog domain each of the caching versions is able to solve one more problem than the control version. The action reordering allows the planner to find the correct solution to the problem using EHC; the control version fails to find a plan via EHC and resorts to best-first search. In the Pipes No-Tankage domain the behaviour of the different versions of the planner is more varied. On the easier problems all three versions of the planner that are doing action reordering solve a number of problems more quickly: most noticeably problems 12, 14 and 15. In problem 15 the action reordering in best first search allows the reordering versions to solve the problem more quickly after resorting to best-first

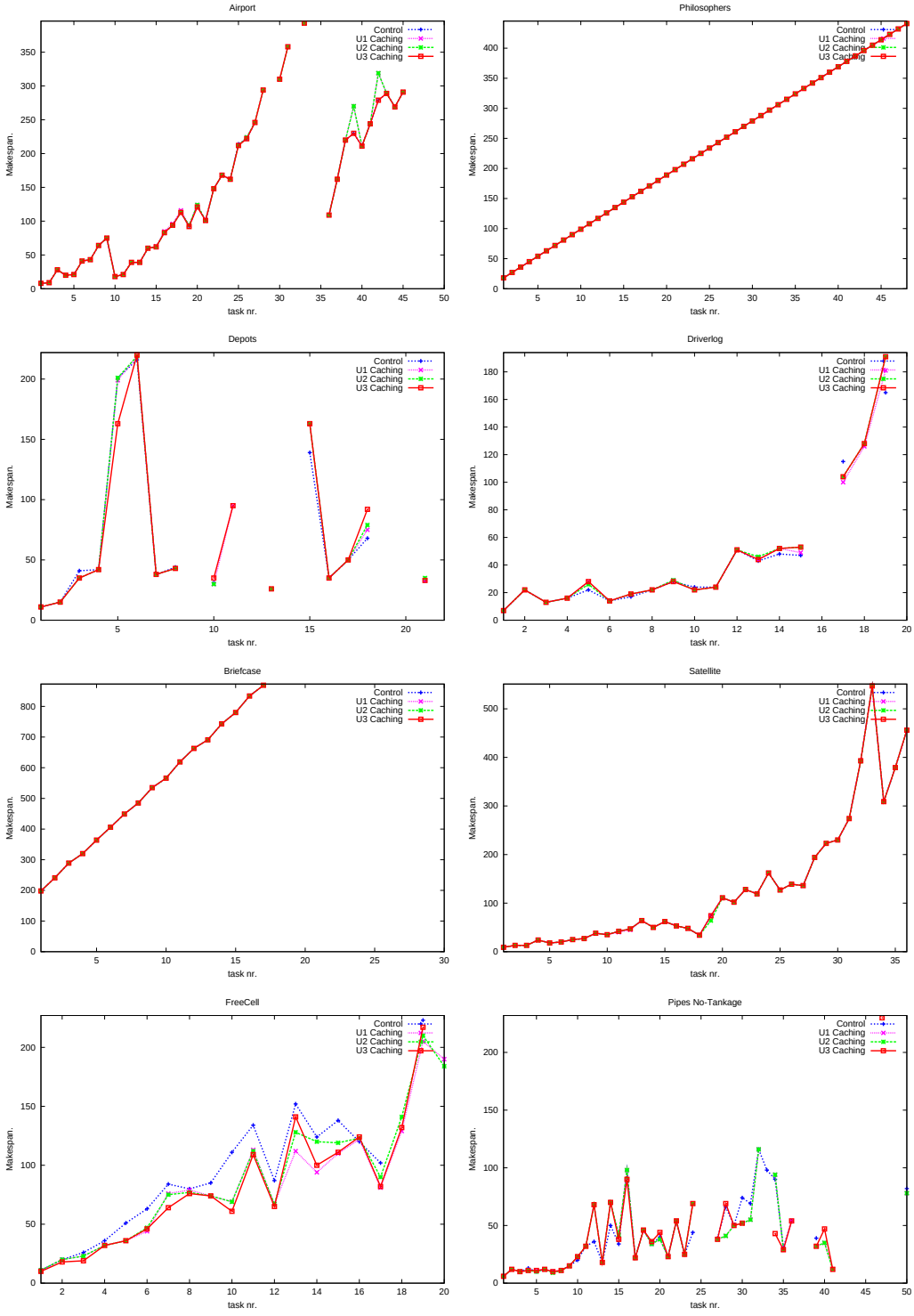


Figure 5.39: Makespan of plans generated in the evaluation domains using the different action reordering update strategies caching reordering data between problems (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	U1 Caching	U2 Caching	U3 Caching
FreeCell	-13.52 *(18)*	13.32 *(18)*	-13.97 *(18)*
Airport	-5.08 *(38)*	-5.01 *(38)*	-1.15 *(38)*
Depots	17.87 *(15)*	17.56 *(15)*	19.71 *(15)*
Philosophers	12.53 *(48)*	12.65 *(48)*	12.66 *(48)*
Driverlog	0.41 *(17)*	-7.94 *(17)*	-7.97 *(17)*
Pipes NT	0.05 *(36)*	-0.49 *(35)*	0.26 *(33)*
Briefcase	7.32 *(17)*	7.19 *(17)*	-36.18 *(17)*
Satellite	45.96 *(36)*	46.45 *(36)*	0.02 *(36)*
Totals	**8.19** ***(225)***	**10.47** ***(224)***	**-3.33** ***(222)***

Table 5.34: Mean of time taken by control version minus time taken using each update strategy with action reordering, caching reordering data. Results are calculated on mutually solved problems

search; whilst in problems 12 and 14 the action reordering versions are able to solve the problem via EHC without the need to resort to best-first search.

Caching the action reordering data in FreeCell leads to the U1 and U2 versions of the planner being able to solve all problems in the FreeCell domain. In problem 18 all versions of the planner resort to best-first search but the cached action reordering data, used in the reordering versions of the planner, allows the problem to be solved using best first search within the time limit. Problem 20 is solved by EHC in the U1 and U2 caching versions of the planner whilst the U3 and control versions remain stuck on a plateau that is not escapable within the time limit. The plans generated by the reordering versions of the planner are shorter in all but one of the problems (see figures 5.39 and 5.35). The planner is able to learn a good ordering strategy on the smaller problems where the trajectory taken is more likely to be closer to the optimal one (as there are fewer possible trajectories). This good ordering strategy is then used when solving the harder problems allowing the planner to generate shorter plans. This is supported by the fact that the makespan of plans is improved when using reordering techniques with no caching but not to the same extent as it is in the case of caching. Shorter

Domain	U1 Caching	U2 Caching	U3 Caching
FreeCell	15.5 *(18)*	11.89 *(18)*	14.56 *(18)*
Airport	-2.39 *(38)*	-2.18 *(38)*	0.11 *(38)*
Depots	-1.6 *(15)*	-2.13 *(15)*	-0.73 *(15)*
Philosophers	0 *(48)*	0 *(48)*	0 *(48)*
Driverlog	-0.88 *(17)*	-1.94 *(17)*	-1.88 *(17)*
Pipes NT	-0.37 *(36)*	-0.34 *(35)*	-0.15 *(33)*
Briefcase	0 *(17)*	0 *(17)*	0 *(17)*
Satellite	0.14 *(36)*	0.31 *(36)*	0 *(36)*
Totals	**1.3** ***(225)***	**0.7** ***(224)***	**1.49** ***(222)***

Table 5.35: Mean of makespan of the solution plan generated by control minus makespan of plan generated using action reordering strategies, caching reordering data. Results are calculated on mutually solved problems.

makespans are a benefit of using macro-actions that is being achieved when using the reordering strategies.

Conclusion

Caching action ordering data can improve coverage on several domains under EHC compared to learning the data on each problem instance and transitively not learning action reordering data. Caching the data allows the planner to build a more accurate representation of the correct ordering strategies based on known successful plans and a much larger collection of successful choices. The U1 caching strategy has greater overall coverage than the U2 and U3 strategies; this is likely to be a consequence of the way the planner naturally considers actions. The U2 and U3 strategies reorder based only on common parameters, the U3 strategy even searching back through the plan to do so; whereas the simpler U1 strategy reorders based on the natural order that the actions appear following each other in the plan. Since it is quite often the case that the next action considered by the planner does not share a parameter with the previous action the U1 strategy is pre-empting the behaviour of the planner more successfully.

Update Strategy	Control vs No Caching	Caching vs No Caching	Total Ordering Significant at 95% ?
U1: (p =)	$1.557 * 10^{-05}$	$5.815 * 10^{-07}$	Yes
sig?	Yes	Yes	
Better	No Caching	Caching	
U2: (p =)	$1.187 * 10^{-05}$	$4.727 * 10^{-07}$	Yes
sig?	Yes	Yes	
Better	No Caching	Caching	
U3: (p =)	0.1203	0.01448	No
sig?	No	Yes	
Better	Control	Caching	

Table 5.36: Significance table for the action reordering strategies: p is the probability that the null hypothesis, that the versions perform the same, cannot be rejected; sig? denotes whether or not the null hypothesis can be rejected with probability ≥ 0.975; Better is the best performing of the two configurations being compared.

Table 5.36 shows the results of performing a Wilcoxon signed-rank significance test on the data generated across all domains. The table shows that over the varied collection of evaluation domains using reordering data can offer significant improvement in time taken to solve problems when using the U1 and U2 strategies. Furthermore, caching action reordering data can allow a significant improvement over learning the data on a per-problem basis using each strategy. For the U1 and U2 versions the final column shows that a total ordering, caching being better than not caching and in turn not caching being better than using no data at all, has been shown to be significant with 95% confidence (since the individual comparisons were shown to be significant with probability greater than $\sqrt{0}.95$). These results confirm the hypotheses that action reordering improves planner performance and that caching the reordering data between problems leads to further improvements. Little difference in makespan is observed: on average the reordering strategies are generating plans approximately 1 step shorter.

5.5.4 Probability-Based Best-First Search on Plateaux

The results in this section show the results of running the planner, using the various update strategies, using best-first search on plateaux, with a heuristic based on the probability that a state is the result of a good chain of actions.

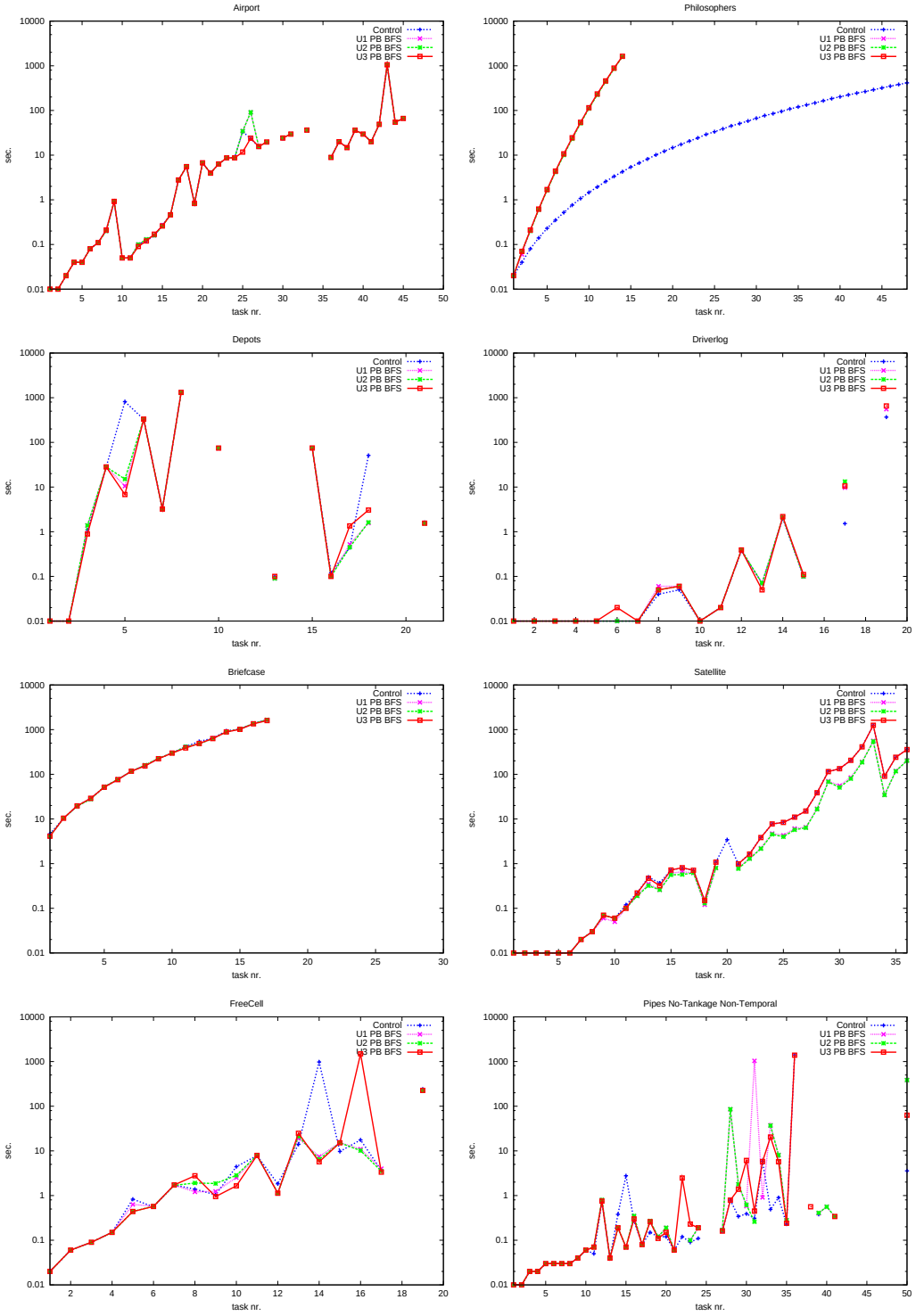


Figure 5.40: Time taken to solve problems in the evaluation domains using probability-based best-first search on plateaux (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	control	U1 Prob. Based BFS	U2 Prob. Based BFS	U3 Prob. Based BFS
FreeCell	18	18	18	18
Airport	38	41	41	41
Depots	15	15	15	15
Philosophers	48	14	14	14
Driverlog	17	17	16	17
Pipes NT	37	36	35	37
Briefcase	17	17	17	17
Satellite	36	35	35	35
Totals	**226**	**193**	**191**	**194**

Table 5.37: Coverage Across Evaluation Domains Using Probability-Based Best-First Search on Plateaux

Hypothesis

Using probability-based best-first search on plateaux will improve planner performance by allowing the planner to find a route off plateaux more efficiently. Plateaux are areas in which the RPG heuristic appears to be uninformative; the learnt action probabilities, however, remain informative in these areas of the search space.

Analysis

The results from this experiment do not provide support for the proposed hypothesis. The overall coverage, shown in table 5.37, is decreased significantly; this is mainly a result of the time taken to solve problems in the Philosophers domain. Best-First search on plateaux has been shown to be much more successful than breadth-first search in this domain [14]. Performing best-first search on plateaux, based on probabilities, prevents the planner from capitalising on the good search guidance offered by the heuristic on plateaux in this domain. The losses can be seen in the Philosophers graph in figure 5.40. The performance degrades to a similar level to that using breadth-first search on plateaux in this domain. This shows that the planner is losing a significant amount of useful search guidance in not following the heuristic; but is not making similar gains by using the probabilistic action reordering data learnt on each problem.

The versions caching action ordering data, shown in figure 5.42, perform consistently better in the Philosophers domain than those not caching action ordering data. The performance is still not as good as the planner using RPG-guided best-

first search on plateaux; the improvement when caching does, however, indicate that using action reordering in this domain can improve planner performance. The more accurate learnt action reordering data collected across many problems is able to guide the planner off plateaux more quickly than in the cases where the planner has learnt only a little knowledge. The effect is particularly pronounced at the beginning of solving problems where the version not caching reordering data have not yet learnt a good ordering. As the patterns of actions required in this domain are regular the U1 and U3 caching strategies are more effective than the U2 strategy. The U1 and U3 strategies will find an action to update at every step and the update will correctly reflect the direction of search that is correct for the planner to take. The U2 strategy does not store as much information in this case when adjacent actions do not share parameters, thus the guidance it provides is slightly weaker (but is still better than the no caching case).

In the Satellite domain the U1 and U2 strategies solve problems more quickly than the U3 and control strategies as shown in table 5.38. There is, however, a decrease in coverage for all the reordering strategies at problem 21. The learnt reordering information leads the planner into a large plateaux which cannot be escaped. On other problems, however, it is clear that the planner is, in general, solving problems in this domain more efficiently, and indeed slightly more quickly than was done without using probability-based best-first search. The Airport domain is the only domain in which the coverage of the control is improved upon when using probability-based best-first search. The coverage improvement in this domain, however, is the same improvement that was seen using the reordering strategies without probability-based best-first search so no additional gain is being made by using the new plateau search strategy.

In the Depots domain the planner is able to solve problem 5 considerably faster using the probability-based best-first search. This is, however, a result of the fact that the technique causes EHC to fail more quickly and the planner to resort to best-first search earlier; rather than providing more effective search guidance. When the planner resorts to best first search the problem is solved very quickly; the control version spends considerably more time attempting to solve the problem via EHC unsuccessfully before resorting to best first search. In problem file 18, however, the planner is able to solve the problem faster in the reordering versions because the action reordering allows the planner find a solution using EHC without the need to resort to exhaustive best-first search. The quality of the plans produced in this problem by EHC is better than that produced by the control using best-first search.

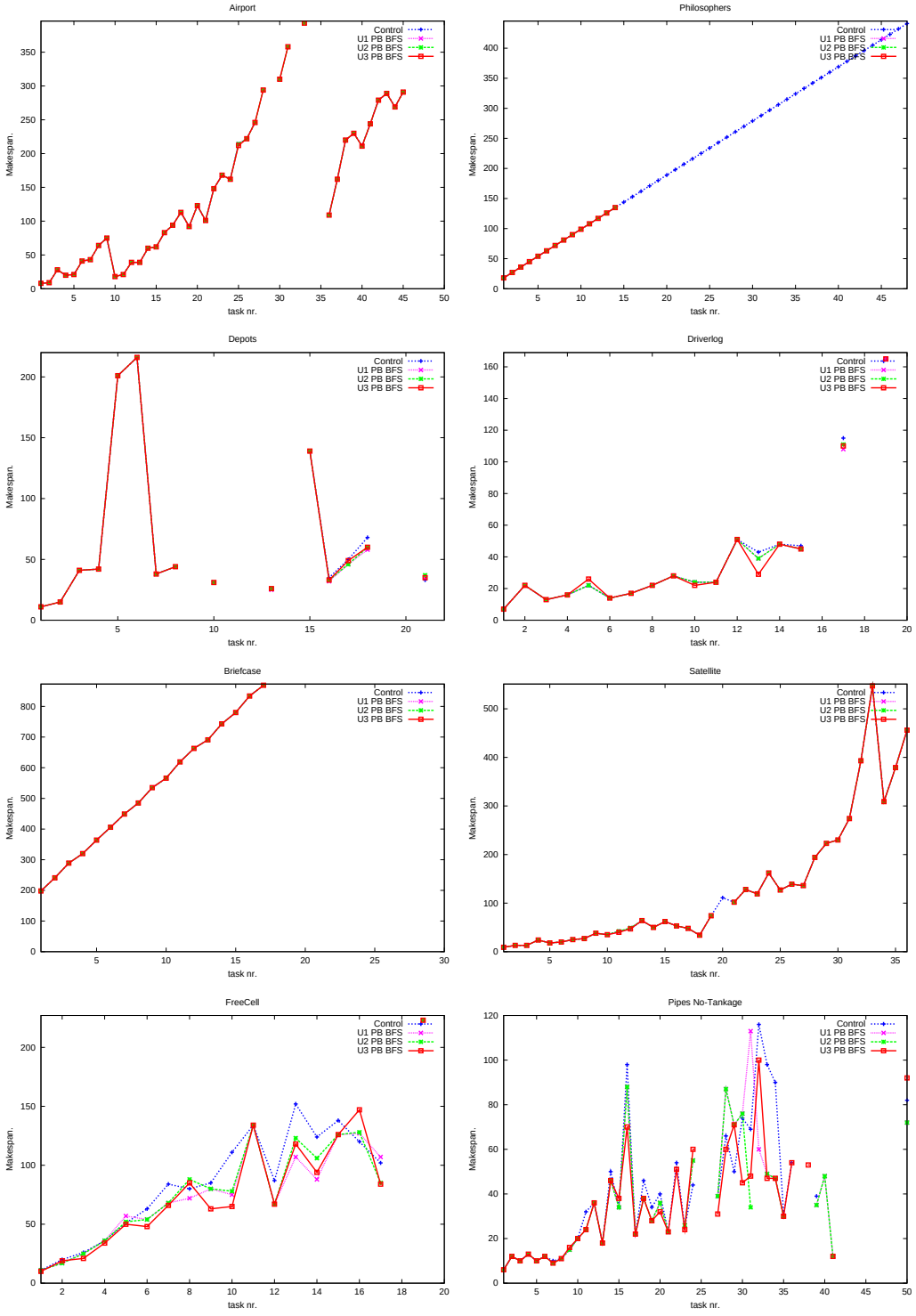


Figure 5.41: Makespan of plans produced in the evaluation domains using Probability-Based Best-First Search on Plateaux (from left to right: Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	**U1** **Prob. Based** **BFS**	**U2** **Prob. Based** **BFS**	**U3** **Prob. Based** **BFS**
FreeCell	54.66 *(18)*	54.72 *(18)*	-27.68 *(18)*
Airport	-1.9 *(38)*	-2.00 *(38)*	0.36 *(38)*
Depots	57.11 *(15)*	56.55 *(15)*	57.12 *(15)*
Philosophers	-242.89 *(14)*	-236.42 *(14)*	-243.43 *(14)*
Driverlog	-11.34 *(17)*	-0.74 *(16)*	-17.43 *(17)*
Pipes NT	-44.01 *(35)*	-14.99 *(34)*	-2.28 *(36)*
Briefcase	6.64 *(17)*	7.28 *(17)*	14.59 *(17)*
Satellite	44.15 *(35)*	45.24 *(35)*	0.31 *(35)*
Totals	**-17.18** ***(189)***	**-11.28** ***(187)***	**-27.31** ***(190)***

Table 5.38: Mean of time taken by control version minus time taken using each update strategy with action reordering using probability-based best-first search on plateaux. Results are calculated on mutually solved problems

The performance in the Driverlog domain is similar on the smaller problems; here the planner does not have to contend with large plateaux[11]. In the larger, more interesting, problems the loss of the guidance from the RPG heuristic is apparent and the performance of the planner is degraded. In the Pipes No-Tankage domain, the slight negative impact on performance seen when using action reordering and RPG-guided best-first search on plateaux is magnified when the guidance of the RPG heuristic is lost.

Observing table 5.38 it appears, at first, that the technique is being very successful in the Briefcase domain; indeed all versions solve problems, on average, more quickly than the control version. Comparing the results in tables 5.38 and 5.32 the improvement gained by doing probability-based best-first search is only slight: most of the improvement being seen is due to the reordering in EHC. The performance in the FreeCell domain again exhibits unpredictability in time taken but an improvement in solution quality. The heuristic does not offer any form of

[11]Large here being number of nodes expanded to escape; rather than simply the length of the exit route.

useful guidance on plateaux in this domain.

Conclusion

The results of the experiment do not support the hypothesis being investigated. The guidance that the planner is getting from the heuristic during best-first search on plateaux is strong in many domains [14]. Searching on plateaux based on probabilities does offer some performance improvement but does not compensate in other domains for the guidance lost by ignoring the heuristic. In domains where the heuristic is not as informative on plateaux some performance improvement can be seen. The makespan of solution plans, shown in figure 5.41 is slightly shorter for the versions using reordering; this is mainly due to improvements in the FreeCell domain where the reordering strategies direct the planner to a shorter path through the search space.

Caching Action Ordering Data

The results generated when caching ordering data using this reordering strategy, shown in figure 5.42, show the same pattern as those not caching reordering data. Coverage, shown in table 5.39 is lost as a result of the loss of heuristic guidance on plateaux. The results do offer an improvement over those not caching reordering data; this is the same phenomenon observed when not using probability-based best-first search. The makespan of the plans generated again shows an overall slight improvement, mainly due to the improvements made in the FreeCell domain. Significance tests are unable to show any significant difference between any probability-based best-first search version and the control version.

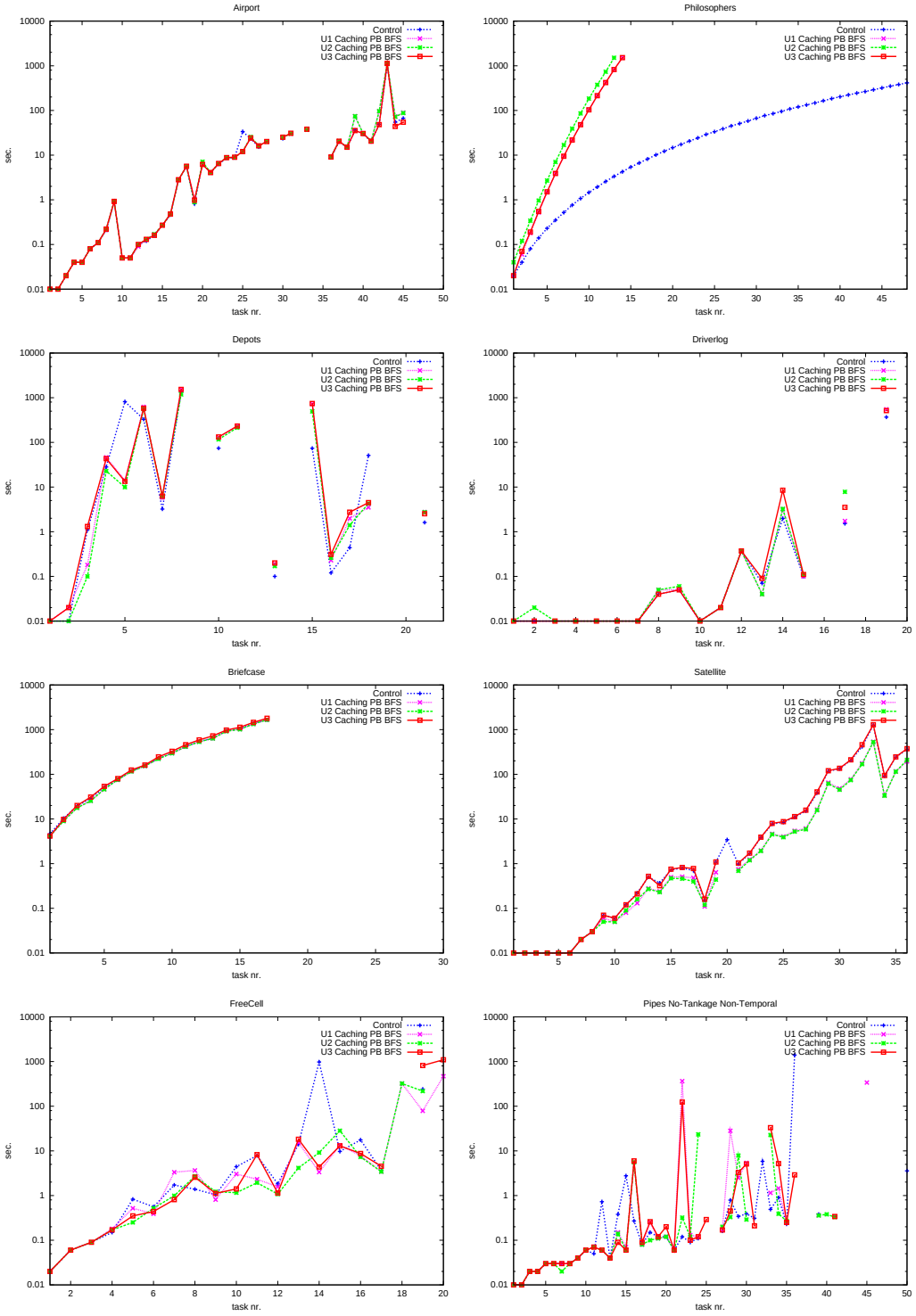


Figure 5.42: Time taken to solve problems in the evaluation domains using Probability-Based Best-First Search on Plateaux and caching action ordering data (from left to right:Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell, Pipes No-Tankage).

Domain	**control**	**U1 Caching Prob. Based BFS**	**U2 Caching Prob. Based BFS**	**U3 Caching Prob. Based BFS**
FreeCell	18	20	19	19
Airport	38	41	41	41
Depots	15	16	16	16
Philosophers	48	14	13	14
Driverlog	17	17	16	17
Pipes NT	37	34	34	35
Briefcase	17	17	17	17
Satellite	36	35	35	35
Totals	**226**	**194**	**191**	**194**

Table 5.39: Coverage Across Evaluation Domains Using Probability-Based Best-First Search on Plateaux Caching Action Reordering Data

Chapter 6

Conclusions

6.1 Summary

This thesis has presented a technique for the online learning of plateau-escaping macro-actions. A number of ways to manage a library of such macro-actions to improve planner performance have been evaluated. Macro-Actions are learnt at points in the search space where heuristic guidance is weak and are saved for re-use at points in the search space where the heuristic cannot offer effective guidance. The macro-actions generated can be used later in solving the same problem and saved for use in future problems.

The simulation of macro-actions through action reordering has also been investigated. A table is maintained recording the number of times a given action has followed another. The actions are ordered for consideration during search according to the number of times they have followed the last action to be added to the plan. Since EHC search selects the first strictly better successor the reordering effectively suggests two-step macro-actions to the planner: following action a immediately consider action b. If action b leads to a better state then the planner will make these decisions without having to consider other possibilities between the two.

6.2 Outcome

A number of experiments have been done to verify the statement of thesis made in section 1.4. The experiments have shown that a library of macro-actions improves the search performance of FF, a state of the art planner, using EHC under the RPG heuristic. This improvement is manifested in three ways: an increased number of problems are solved; time taken to solve problems is signif-

icantly reduced and solutions of significantly higher quality are produced. This is noticeable across a wide range of domains with very different properties under the RPG heuristic.

The investigation began with experiments to evaluate the use of plateau-escaping macro-actions in planning. The findings are as follows:

- Applying plateau-escaping macro-actions after all other actions results in the best overall planner performance: allowing performance improvement whilst minimising overheads.
- Plateau-escaping macro-actions significantly improve solution quality allowing the planner to produce considerably shorter solutions to problems.
- Macro-actions of various lengths improve search performance; not only those of length 2, as is often suggested.

Efficient search-time pruning allows the planner to search keeping all macro-actions in the library and perform significantly better than the versions not caching macro-actions. Indeed, search time pruning is, alone, sufficient as a library management strategy. The search time pruning was shown to offer significant performance improvement, in terms of time taken to solve mutually solved problems, over not using search time pruning; further the pruning version solved 44 more problems than the no pruning version. Search time pruning takes the form of helpful macro-action pruning, only considering macro-actions whose first step is a helpful action; symmetric action pruning, removing redundant identical choices; and considering macro-actions only when the planner reaches a plateau.

The best performing version of the planner overall was that using the survival of the fittest strategy and keeping the top 10 macro-actions. The top 10 version solved 247 of the problems in the evaluation suite; compared to the 225 solved by the no macro-actions version. Further, it found solutions to mutually solved problems on average 67 seconds faster than the control, representing a statistically significant performance improvement. The improved performance of the top 10 version relative to the keep all version of the planner demonstrates that additional library management offers significant performance gains. Further, it was shown that the number of times a macro-action appears in a solution plan is a good indicator of whether it is likely to be useful again. Other library management strategies have been investigated, these all gave rise to performance and coverage improvement over not using macro-actions, and not caching macro-actions, in most configurations.

Action reordering was shown to have a positive impact on coverage and significantly reduce time taken to solve problems across the range of evaluation domains. The action reordering strategies did not give rise to a performance improvement that was as great as that achieved using macro-actions; but did improve performance and coverage without the negative impact observed in some problems when using macro-actions. The benefits of action reordering do not necessarily occur in domains in which macro-actions are the most useful; in some domains, action reordering is helpful when macro-actions are not necessarily so. Extending the use of action reordering to use a probability-based best-first search on plateaux was not successful as the negative impact of the loss of the RPG heuristic guidance on plateaux outweighed the benefits.

Overall this thesis has shown that online macro-action generation techniques offer significant improvement in the performance of FF, a state-of-the-art planner. The performance benefits gained are a significant reduction in time taken to solve mutually solved problems, a significant improvement in solution quality and an increase in the number of problems solved. Action reordering techniques have also been shown to offer significant performance improvement, when used in the same planner, without compromising solution quality.

6.3 Future Work

The work presented in this thesis opens up many possibilities for future developments. This section details potential future directions to extend the work on plateau-escaping macro-actions and the simulation of macro-actions through action reordering.

Distinguishing Between Heuristic-Improving and Search-Improving Macro-Actions

Macro-actions can assist a planner in two ways. The first is allowing the application of multiple action steps with one choice point (thus decreasing the depth to which the search tree must be explored); the second is providing extra search guidance through being included in the calculation of the RPG heuristic value. Marvin does not use macro-actions in the heuristic computation in the default configuration: this is because including many macro-actions in the computation step makes the calculation of the heuristic much more expensive. It is, however, possible that including a small number of short macro-actions in the heuristic calculation could greatly improve the accuracy of the heuristic estimate obtained;

thus improving the efficiency of search. This phenomenon has been observed by Adi Botea[1].

Zimmerman and Kambhampati [73] note that despite many of the recent steps forward made in heuristic-search planning being attributed to the heuristic, very little of the research in learning in AI planning has been directed towards learning to improve the heuristic. Macro-Actions can, however, increase the quality of the heuristic value by forcing the planner to select two interlinked actions for the relaxed plan, when normally only one of these actions may have been added. Take, for example, the macro-action drop-pickup, where the drop action achieves a predicate hands-free that is then required and deleted by the pickup action. In a planning problem where the hands-free predicate appears in the initial state the relaxed plan can use the action pickup at any point: the handsfree predicate will never be deleted in the initial state. Using the pickup action without a drop action will result in the relaxed plan becoming increasingly inaccurate every time the pickup action is applied. Adding the drop-pickup macro-action to the relaxed planning graph (as if it were a single action) will introduce the correct action sequence without the need for a specialist relaxed-plan extraction algorithm. Macro-actions will generally be selected in preference to other achievers in the graph as the standard algorithm selects the earliest possible achiever. Macro-actions represent multiple actions allowing predicates to be achieved at earlier layers. The example given can apply to many actions sequences which would normally only be applicable as a complete unit.

Extending the Use of Plateau-Escaping Macro-Actions To Other Planning Strategies

The investigation can be extended further to explore the wider applicability of plateau-escaping macro-actions. This includes use of plateau-escaping macro-actions, generated under the RPG heuristic, in planners that do not use this heuristic. The motivation for this is that the core hard part of a problem exists when solving it using any approach, and that part does not lie where the relaxed planning graph can accurately model the problem polynomially. Indeed, it was observed in section 5.4 that many of the macro-actions generated under the RPG heuristic were also generated under the CG heuristic.

The extension of plateaux-escaping macro-actions to other planning paradigms can also be considered. For example, LPG reaches plateaux in search spaces just as Marvin does. As LPG is searching using a very different structure,

[1]Personal communication: research seminar University of Strathclyde, UK.

a planning graph with selected actions, plateau-escaping macro-actions would take a very different form in this search.

Extraction of Plateau-Escaping Macro-Actions from Optimal Plans

The improvement in solution quality that can be observed by using plateau-escaping macro-actions has been shown in section 5.2.1. Extraction of macro-actions from optimal plans could potentially further enhance the benefits gained by allowing segments of optimal plans to be used in solving harder problems. This may allow even greater improvements in solution quality to be seen.

Action Reordering Techniques

The work done on action ordering strategies can be further extended to allow greater performance enhancements. In order to attempt to gain a greater makespan improvement ordering data could be extracted from optimal plans, in the same way macro-actions could be.

Currently the action reordering only simulates macro-actions of length 2 as only a single previous action is considered. Macro-actions of other lengths could be simulated, by the consideration of the preceding n actions, rather than just the one immediately preceding action. The table of action reordering data would, however, grow rather large as each pairwise combination of actions would have to be considered to precede each action. A strategy to select promising combinations of actions, and only consider these, would therefore be required.

Bibliography

[1] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[2] C. Backstrom and B. Nebel. Complexity results for SAS+ planning, 1995.

[3] Paul Benjamin, Leo Dorst, Indur Mandhyan, and Madeleine Rosar. An introduction to the decomposition of task representations in autonomous systems. In *Chance of Representation and Inductive Bias*, pages 125–146, 1990.

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the International Joint Conference on Artificial Inteligence (IJCAI-95)*, 1995.

[5] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[6] Adi Botea, Martin Müller, and Jonathan Schaeffer. Using component abstraction for automatic generation of macro-actions. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 181–190, 2004.

[7] Adi Botea, Martin Müller, and Jonathan Schaeffer. Learning partial-order macros from solutions. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 05)*, pages 231–240, 2005.

[8] Adi Botea, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[9] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[10] Jaime Carbonell, Craig Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In *Architectures for Artificial Intelligence*, pages 241–278, 1991.

[11] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[12] A. I. Coles and A. J. Smith. Marvin: Macro-actions from reduced versions of the instance. IPC4 Booklet, ICAPS 2004, June 2004.

[13] Andrew Coles. *Heuristics and Metaheuristics in Forward-Chaining Planning.* PhD thesis, University of Strathclyde, 2006.

[14] Andrew Coles and Amanda Smith. MARVIN: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 2006. Accepted for publication.

[15] C. Dawson and L. Siklossy. The role of preprocessing in problem solving systems. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence, IJCAI 77*, pages 465–471, Cambridge, MA, 1977.

[16] Minh B. Do and Subbarao Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of ECP 2001*, 2001.

[17] Rocio Duran. Integrating macro-operators and control rules learning. Doctoral Consortium, ICAPS 2006, June 2006.

[18] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report No. 195, Institut fr Informatik, December 2003.

[19] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[20] R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620, 1971.

[21] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[22] M. Fox and D. Long. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[23] M. Fox, D. Long, and J. Porteous. Abstaction-based action ordering in planning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[24] M. Fox and D.P. Long. The detection and exploitation of symmetry in planning problems. In *In Proccedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 956–961, 1999.

[25] A. Gerevini and D. Long. Plan constraints and preference in PDDL 3. Technical Report Technical Report RT-2005-08-47, Dep. di Elettronica per l'Automazione, Universita di Brescia, Italy, 2005.

[26] Alfonso Gerevini and Ivan Serina. LPG: a planner based on local search for planning graphs. In *Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02) Toulouse, France*. AAAI Press, 2002.

[27] Keith Halsey. *CRIKEY! It's Co-ordination in Temporal Planning: Minimising Essential Planner–Scheduler Communication in Temporal Planning.* PhD thesis, University of Durham, 2004.

[28] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 04)*, 2004.

[29] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[30] Malte Helmert. New complexity results for classical planning benchmarks. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 52–61, 2006.

[31] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519 – 579, 2005.

[32] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[33] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 453–458, 2001.

[34] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, 2002. 379-387.

[35] Jörg Hoffmann. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.

[36] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[37] Jörg Hoffmann and Bernhard Nebel. What makes the difference between HSP and FF? In *Proceedings IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, 2001.

[38] Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.

[39] Glenn A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317, 1989.

[40] B. Kautz, H. Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden (IJCAI '99)*, 1999.

[41] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[42] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *ECP*, pages 273–285, 1997.

[43] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[44] R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[45] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[46] Jonas Kvarnström and Martin Magnusson. TALplanner in the third international planning competition: Extensions and control rules. *Journal of Artificial Intelligence Research (JAIR)*, 20:343–377, 2003.

[47] J. Levine and D. Humphreys. Learning action strategeis for planning domains using genetic programming. In J. Porteous, editor, *Proceedings of the 23rd UK Planning and Scheduling SIG*, 2003.

[48] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[49] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research (JAIR)*, 20:1–59, 2003.

[50] Mario Martin and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.

[51] T. L. McCluskey. Combining weak learning heuristics in general problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI '87)*, 1987.

[52] T. L. McCluskey. *Experience Driven Heuristic Acquisition in General Problem Solvers*. PhD thesis, The City University, London, 1988.

[53] T. L. McCluskey and J. M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95(1):1–65, 1997.

[54] Drew McDermott. The 1998 AI planning systems competition. In *AI Magazine 2*, pages 35–55, 2000.

[55] Drew V. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[56] S. Minton. Selectively generalizing plans for problem-solving. In *Proceedings of the ninth International Joint Conference on Artificial Intelligence, IJCAI '85)*, 1985.

[57] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–973, 1999.

[58] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, 2003.

[59] M.A.H. Newton, J. Levine, and M. Fox. Genetically evolved macro-actions in A.I. planning problems. In A. Tuson, editor, *Proceedings of the 24th UK Planning and Scheduling SIG*, 2005.

[60] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 459–466, 2001.

[61] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[62] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[63] J. S. Penberthy and D. S. Weld. Ucpop: A sound, complete, partial order planner for adl. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 103–114. Kaufmann, San Mateo, CA, 1992.

[64] J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pages 32–40, 2003.

[65] Kambhampati. S. Are We Comparing Dana and Fahiem or SHOP and TLPlan? A critique of the Knowledge-Based Planning Track at IPC. In *ICAPS 03 Workshop on the Competition*, 2003.

[66] E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214, 1975.

[67] A. Tate. Generating project networks. In *Proceedings of the fifth International Joint Conference on Artificial Intelligence, IJCAI 77*, pages 888–893, 1977.

[68] M. Veslo and J. Carbonell. Toward scaling up machine learning: A case study with derivational analogy. In S. Minton, editor, *Machine Learning Methods for Planning*, pages 233–272. Morgan Kaufmann, San Mateo, CA, 1993.

[69] Vincent Vidal. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 150–160, 2004.

[70] Vincent Vidal. The YAHSP planning system: Forward heuristic search with lookahead plan analysis. IPC4 Booklet, ICAPS 2004, 2004.

[71] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[72] Håkan L. S. Younes and Michael L. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania., 2004.

[73] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2):73–96, 2003.