Online Identification of Useful Macro-Actions for Planning

Andrew Coles, Maria Fox and Amanda Smith
Department of Computer and Information Sciences,
University of Strathclyde,
26 Richmond Street,
Glasgow, G1 1XH
email: `firstname.lastname@cis.strath.ac.uk`

Abstract

This paper explores issues encountered when performing on-line management of large collections of macro-actions generated for use in planning. Existing approaches to managing collections of macro-actions are designed for use with off-line macro-action learning, pruning candidate macro-actions on the basis of their effect on the performance of the planner on small training problems. In this paper we introduce macro-action pruning techniques based on properties of macro-actions that can be discovered online, whilst solving only the problems we are interested in. In doing so, we remove the requirement for additional training problems and offline filtering. We also show how search-time pruning techniques allow the planner to scale well to managing large collections of macro-actions. Further, we discuss the properties of macro-actions that allow the online identification of those that are likely to be useful in search. Finally, we present results to demonstrate that a library of macro-actions managed using the techniques described can give rise to a significant performance improvement across a collection of domains with varied structure.

1 Introduction

Previous approaches to generating macro-actions for planning use offline learning techniques to select good macro-actions from a collection of potential macro-actions. Offline learning approaches use some training problems to rank and filter macro-actions in order to decide which macro-actions should be kept, based on the success of solving these problems. A recent competitor in the Fourth International Planning Competition (IPC4), Macro-FF (Botea, Müller, & Schaeffer 2005), extracts macro-actions from solution plans and uses training problems to identify the most useful. Another recent planner (Newton, Levine, & Fox 2005) uses genetic programming to generate macro-actions again using, as part of the fitness function evaluation, a collection of small training problems. In common between both of these planners is the need to perform an offline learning step to before planning can take place in a given domain.

These supervised learning techniques work by solving training problems with and without using a given feature,

in this case a macro-action, and comparing the results obtained to decide whether or not it is beneficial. In this paper, however, we consider the problem of managing a large collection of macro-actions *online* without the need to solve training problems. We build on the planner Marvin (Coles & Smith 2007), another participant in IPC4, which introduced a mechanism for inferring macro-actions online whilst solving each problem. These macro-actions were not, however, available for use when solving other problems in the domain. In this paper we present a technique that builds on this online macro-action inference, keeping libraries of macro-actions for use on future problems, and introducing library management strategies. This remains an online technique, with all reasoning being done during planning and included in the time taken to solve the planning problems. In doing so, no additional problems are solved for training, only each problem we are required to solve, once.

2 Background

The use of macro-actions in planning has been widely explored. Macro-actions consist of an ordered sequence of action schemata 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 it might be good sequence to consider. In the context of planning problems, *unit actions* are considered to be those that are specified in the domain; and *macro-actions* are sequences of one or more of these. Ground actions are actions whose parameters have been bound to entities.

In this work, we are concerned with macro-actions which are inferred from sequences of ground unit actions. Defining pre_X, add_X and del_X to be sets of preconditions, add effects and delete effects of a ground action X, we can define the process of inferring macro-actions from unit actions recursively. A ground macro-action of length 1 corresponds to any ground unit action. A ground action a_1 can be added to the end of an existing ground macro-action a_m if, and only if, $del\_a_m \cap pre\_a_1 = \emptyset$. The result of adding a_1 to the end of a_m is a new ground macro-action a_n where:

$pre\_a_n = pre\_a_m \cup (pre\_a_1 \setminus add\_a_m)$
$add\_a_n = add\_a_m \cup add\_a_1$
$del\_a_n = (del\_a\_m \cup del\_a_1) \setminus add\_a_1$

A ground macro-action can then be used to build a macro-

action schema by replacing the bound variables of preconditions and effects with placeholder parameters. Importantly, if a given entity appears in several preconditions or effects, it gives rise to only one parameter. In doing so, the relationships between the steps of macro-actions are preserved.

2.1 Online Macro-Action Inference

In the Fourth International Planning Competition (IPC4) the planner Marvin (Coles & Smith 2007) introduced a mechanism for the online learning of macro-actions during search. Marvin is a development of the ideas introduced in the FF (Hoffmann & Nebel 2001), in particular introducing the use and inference of macro-actions into search. In Marvin, as in FF, search follows a forward-chaining approach to planning, obtaining heuristic guidance by building a Relaxed Planning Graph (RPG) in each state: a Graph-Plan (Blum & Furst 1995) planning graph, in which the delete effects of actions are ignored. A solution extracted from the RPG is used to provide heuristic guidance in two ways. First, the length of the solution is used as a goal-distance estimate. Second, *helpful-action pruning* is used, where the choice of actions applicable in a state is restricted to those which share an add effect with those chosen from the first action layer in the relaxed solution.

The search algorithm used in Marvin and FF is a variant of hill-climbing, known as Enforced Hill Climbing (EHC). EHC can be seen as a hybrid of local-search hill-climbing with exhaustive search used to escape local minima (breadth-first search in FF, best-first search in Marvin). In EHC, when expanding a state to generate its successors, each is evaluated in turn. If a successor has a better heuristic value than the global best state, it is immediately chosen and EHC proceeds from there. As such, EHC does not follow a steepest-descent approach: the first successor found with a heuristic value better than the previous global optimum is always chosen for expansion. In the absence of a successor with a heuristic value better than the global optimum, exhaustive search is used until such a state is found.

Areas in which no strictly better state is found amongst the immediate successors, and in which the planner is performing exhaustive search, are referred to as *plateaux*[1]. The majority of the search time when using EHC is spent searching these regions in which the heuristic can offer no guidance. Sequences of actions used to transition from a state at the start of a plateaux to a state off the plateau are, therefore, expensive to compute. Marvin creates macro-actions from these sequences of actions for use later in search memoising the action sequence for use on future plateaux; full details of this process can be found in (Coles & Smith 2007). In this work, we explore storing these macro-actions, not only for use during search to solve a given problem, but for use on future problems in the same domain. Storing macro-actions for use on future problems introduces new challenges: very large collections of macro-actions are generated and the effective management of these is essential.

---

[1]In this work, the term plateaux is used to refer to both plateaux and saddle points

2.2 Problems Associated with Large Collections of Macro-Actions

Using macro-actions when planning gives the potential for increased efficiency, 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: the increased branching factor at each action choice point in the search. In addition to considering all possible instantiations of the unit actions, instantiations of macro-actions must also be considered. In the worst case, the number of instantiations of an action is exponential in the number of parameters. Macro-action schema derived from two or more actions include the necessary parameters of their constituent actions. As such, in general, a macro-action schema will allow the instantiation of many ground-macro-actions.

For a planning problem with solution length l and number of possible unit action instantiations (branching factor) b the number of nodes explored to solve the problem is $O(b^l)$. If m macro-actions schema are available, leading to the instantiation of n ground macro-actions, this rises to $O((b+n)^l)$. It is, however, important to note that if a ground macro-action leads to a solution then the depth l to which the search space needs to be explored (i.e. the plan length) is decreased: a macro-action of length k (where $k > 1$) reduces the number of nodes to $O((b + n)^{l-(k-1)})$.

Further problems arise regarding the effects of macro-action use on 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. As such, 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.

3 Maintaining The Library of Macro-Actions

Having solved a planning problem and generated macro-actions during the process, it would be useful if this information was available in solving subsequent problems. In doing so, however, we risk suffering from the problems associated with using large collections of macro-actions. Our approach to dealing with these issues is based around extracting features from each of the macro-actions generated, and using these to assess which macro-actions will be useful for consideration during search and which should not be used. To support high-level reasoning about macro-actions, we store macro-action schemata in a non-compiled form; that is, rather than being stored as single actions with preconditions and effects, they are stored as a sequence of unit actions from the domain, together with parameter constraints indicating which parameters are shared between which unit actions. Storing macro-actions in this way allows us to reason about them more accurately, referring to the constituent actions, rather than being forced to treat them as a black box.

As well as the details of the unit actions within the macro-actions, macro-actions have several features that can be identified online, during problem solving. These features can be stored in the macro-action library, along with the macro-actions themselves, and used to identify which are the most likely to be useful. Pruning on the basis of these features introduces the risk of discarding macro-actions that would have been useful, so these features must be used intelligently to identify good macro-actions. In this section we identify the features we consider and how they are used.

3.1 Usage Count

Within a domain, macro-actions that have successfully been used in solving many problems are likely to be useful in solving future problems. Offline learning approaches often use this premise to determine the usefulness of macro-actions using small training problems. The advantage that offline techniques have is that they can solve the problem once with the macro-action schema available, and once without. In doing so, it can be seen not only whether the macro-action has been used, but also if using it has reduced the time taken to solve the problem. Using an online, unsupervised, learning technique we do not know how quickly the problem would have been solved without the macro-action; instead we must rely on an assumption that the use of a macro-action in planning will have reduced the amount of time taken to solve the problem. Whether or not this assumption holds will determine the success of this strategy.

To collect usage count statistics during search, we count how many times an instantiation of each macro-action has been used in a solution plan. To prune macro-actions with this, we introduce a *survival of the fittest* library pruning strategy. Here, only the n most-used macro-actions from the library are kept; the remainder are pruned. Then, only these n macro-actions are available when planning, along with any new ones inferred during search. We also consider a variant on this, a *roulette selection strategy*, in which at each point in search where macro-actions are considered, n macro-actions are selected from the library, probabilistically, based on their usage counts. The size of the roulette wheel segment for each macro-action is its usage count plus one (to ensure that macro-actions that have not yet been used still have a chance of being selected). The motivation for the roulette selection strategy is that macro-actions that have not proven useful immediately remain in the library rather than being discarded, thereby giving newer macro-actions an increased chance of survival.

To update usage counts after a problem has been solved, we post-process the plan to identify plateaux in the search landscape. The plateau-escaping sequences are then extracted and converted into macro-actions. Each macro-action is then checked against the macro-action library to see if duplicates an existing macro-action. If it does, the usage count of the existing macro-action is is incremented; if not, the new macro-action is added to the library.

3.2 Number of Problems Since Last Use

Another feature of macro-actions that can be maintained is the number of problems that have been solved since a given macro-action was last used in a solution plan. The *time out* pruning strategy works with this feature, removing macro-actions from the library that have not been used in solving the n most recent problems. The motivation behind this technique is to maintain only the macro-actions used in solving the most recent problems, so that if the structure of the problems changes over time, more recently generated, more relevant, macro-actions can be maintained. As when updating usage counts, we use solution plan post-processing to identify when macro-actions have been used.

3.3 Instantiation Count

The number of times a macro-action has been instantiated is an important factor in determining its overall contribution to search time. If a macro-action is instantiated many times then its cost in terms of increasing the branching factor is high; if it is instantiated few times, it has a low cost. What is perhaps more important, however, is not the number of times a macro-action is instantiated, but the ratio between that and the number of times it is used in a solution plan. A macro-action that is instantiated many times but used in many plans does increase the branching factor, but is being very useful in search; whereas one instantiated equally many times, but rarely used, is increasing the branching factor without giving the benefit of reduced search effort. Similarly, a macro-action that is instantiated only a few times, but used each time it is instantiated, is potentially valuable; whereas one that is instantiated occasionally but never used, although not greatly damaging to performance, is clearly not being useful. The *instantiation versus usage count* pruning strategy works on this basis, pruning macro-actions from the library when the usage count divided by the instantiation count is below a specified threshold, θ.

In Macro-FF, a similar strategy is used, once, after searching to solve five problems in a given domain. Our strategy makes this pruning truly dynamic and online, being invoked after solving every problem, rather than once after five. In this work, macro-actions are generated online, and on any problem with one or more plateaux. As such, it is necessary to frequently prune based on instantiation and usage counts. In Macro-FF, however, no new macro-actions are generated during search, so applying this pruning after each problem was solved would result in the library size decreasing monotonically with no potential for macro-actions to be replaced.

3.4 Length of Macro-Actions

Length is often used as a criterion for selecting or pruning macro-actions, with many planning systems pruning macro-actions above length 2. Short macro-actions are often favoured as they generally have fewer parameters, and hence fewer possible instantiations. As such, they give rise to a smaller increase in branching factor. Another motivation for selecting shorter macro-actions is to minimise the potential impact on solution plan length: if a long macro-actions is used in a solution plan, but without it fewer unit actions would have been used for the same purpose, then a longer plan is generated. Using longer macro-actions does, however, have advantages: the application of each macro-

action takes the planner potentially more steps towards the goal at any one time.

Our system does not make any strict requirements regarding macro-action length; as such, we can impose additional constraints on macro-action lengths to determine whether keeping only the shortest macro-actions is the best policy. As macro-actions are stored in a format that lists their constituent actions, we have a record of the length of all macro-actions. We consider the strategy of pruning macro-actions based on their length, maintaining only those actions that are shorter than a specified threshold. We also investigate how often macro-actions of different lengths are used in solution plans to determine whether there is a link between the length of a macro-action and usefulness.

4 Search Time Pruning Techniques

We use two search-time mechanisms to minimise the potential negative impact of macro-actions on planning: helpful macro-action pruning and macro-action ordering. Intelligently deciding *when* to apply macro-actions, and *which* macro-actions to apply, can prevent search from becoming more difficult due to many irrelevant choices being posed.

4.1 Helpful Macro-Action Pruning

The RPG heuristic, as used in FF (Hoffmann & Nebel 2001) and Marvin, is based on solving a relaxed version of the planning problem—one in which the delete effects of actions are ignored. To evaluate a state, a 'relaxed' plan to solve the relaxed problem is built from the state to the goal, and the length of the relaxed plan gives a heuristic goal distance estimate. Additionally, the relaxed plan is used to compute what are referred to as *helpful actions*. Helpful actions are those that achieve any effect of the actions at the first time step in relaxed plan. Only these helpful actions are during performing EHC search.

Helpful action pruning can be extended to macro-actions to dramatically reduce the number of macro-actions to be considered when expanding a state. *Helpful macro-action pruning* uses the first unit action step of an instantiated macro-action to decide whether it is helpful. In this work, we use the standard definition of helpful actions to prune unit actions (those actions found in the domain definition). However, we use a more strict definition of helpful actions for the purposes of macro-action pruning. We consider only those actions that actually appear at the first step of the relaxed plan as helpful, without additionally including those that achieve the same effects.

If the first unit action that makes up an instantiated macro-action is a helpful action (according to our stricter definition) then the macro-action is considered to be helpful. In the case where a macro-action contains concurrency, and several actions are present in the first time step, all of these actions must fulfil the strict helpful actions definition for the macro-action to be considered helpful. Our helpful macro-action pruning differs from that in Macro-FF (Botea, Müller, & Schaeffer 2005) in two ways. First, we used a revised definition of helpful. Second, in Macro-FF, multiple subsequent steps of macro-actions must match against the relaxed plan; all of our pruning is done based on the first time step.

To avoid having to instantiate a large number of potentially non-helpful macro-actions before planning, we only instantiate macro-actions lazily, on an as-needed basis. When producing instances of a macro-action for application in a given state, we first loop over the actions in its first time step, referring to the helpful-actions of the state to ensure that only helpful macro-actions are instantiated. For subsequent time steps of the macro-action, we only allow bindings that give rise to applicable action sequences. This lazy instantiation allows us to reason with larger collections of macro-actions, without the great expense of instantiating all macro-actions before planning.

4.2 Macro-Action Ordering

Ordering macro-actions correctly with respect to the unit actions is critical for search efficiency. Ordering macro-actions before unit actions will be beneficial if we have a small collection of very useful macro-actions that almost always represent the correct search path to take. Ordering them after will be beneficial if they are useful only sometimes. In the case of Marvin, where EHC is used, ordering macro-actions after all unit actions (the default configuration) corresponds to only applying macro-actions on plateaux: if a unit action was available to lead to a strictly better state, it would have been greedily selected and added to the plan. As macro-actions are inferred on plateaux in Marvin, the effect is that macro-actions are only considered in areas of the search landscape with a topology similar to that in which they were first used.

These two extreme cases of macro-action ordering do not represent the only possible strategies. It is possible to order any number of macro-actions to be considered before the unit actions found in the domain. In the generated library it may often be the case that, for example, the two most used macro-actions almost always reduce search time when applied, whereas the remainder are better if only considered for application when no other action has lead to a strictly better state. Ordering macro-actions after the unit actions minimises their impact on search performance; whilst ordering them before unit actions maximises their potential for improving search performance. In its default configuration, Marvin orders macro-actions to be after unit-actions, thereby attempting to use previous plateau-escaping sequences only on plateaux. In contrast, Macro-FF has a series of macro-actions generated offline, whose performance has been analysed using several offline runs of the planner on small training problems. Knowing that these macro-actions are likely to give rise to a performance enhancement if applied, they are ordered before the unit actions.

Macro-action ordering becomes increasingly important as larger macro-action libraries are generated. In the evaluation section we investigate the scalability of the planner using our macro-action ordering strategy and investigate considering some macro-actions before the unit actions.

5 Results

In this section we highlight some results from our experiments done to identify the best way to manage a large collection of macro-actions. The tests have been performed across

a range of domains with differing properties, taken from recent planning competitions (Long & Fox 2003) (Hoffmann & Edelkamp 2005): Airport, Philosophers, Depots, Driverlog, Briefcase, Satellite, FreeCell and Pipes-NoTankage; a total of 266 problems. Further results, with a breakdown on a domain-by-domain basis for each configuration, can be found in (Smith 2007). The problems are considered in numerical order, as specified in the planning competitions. This has a general trend towards easier problems being considered first and more difficult problems later, although this ordering is often not entirely correct. Experiments were performed on a computer with 1Gb of RAM and a 3.4GHz Pentium 4 processor. Each planner invocation is restricted to 30 minutes of CPU time and 800MB of memory.

Mean time improvement figures presented are the mean of the differences in the time taken by a pair of configurations to solve each problem. That is, for pair of configurations B and A, the mean time improvement of B over A is given by $(1/n) \sum_{i=1..n} (Time(A_i) - Time(B_i))$, where n is the number mutually solved problems and $Time(A_i)$ and $Time(B_i)$ are the time taken by configurations A and B, respectively, to solve problem i. Times are only included for mutually solved problems. Makespan improvement is calculated in the same way using makespan data. All statistical testing is performed using the Wilcoxon Signed Rank Test to 95% confidence, unless otherwise stated. This test was selected as it does not rely on the assumptions of normally distributed data imposed by more conventional Z- and t-tests.

5.1 Helpful Macro-Action Pruning

The first set of experiments are concerned with evaluating our search time pruning techniques. For the purposes of this experiment we keep all macro-actions generated in the macro-action library, sorted by usage count so macro-actions are considered in the order most-used to least-used. The results generated with helpful macro-action pruning enabled can be seen in Figure 1, labelled 'Keep All'. The 'No Search Time Pruning' configuration, in which helpful macro-action pruning is disabled, solves only 73.7% of the evaluation problems; compared to the 90.2% solved by 'Keep All' configuration. Furthermore, the mean time improvement achieved by using helpful action pruning is 110.19 seconds; that is, mutually solved problems are solved on average 110.19 seconds faster. The Wilcoxon Signed Rank Test also shows there to be a significant reduction in time taken to solve problems.

The helpful macro-action pruning is of vital importance when macro-action libraries are maintained, with the results demonstrating that our helpful macro-action pruning technique allows the planner to scale to reasoning with a large library of macro-actions. Even without other library management strategies to try and eliminate poor quality macro-actions, the planner exhibits better performance when keeping all macro-actions than the 'No Caching' (macro-actions discarded after each problem) and 'No Macro-Actions' configurations. Disabling the helpful macro-action pruning, however, reduces the performance of 'Keep All' to below the baseline 'No Macro-Actions' configuration. The impact
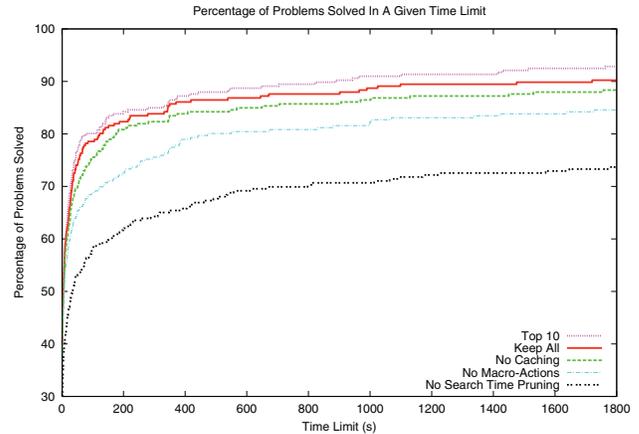

Figure 1: Percentage of Problems Solved in a Given Time Limit by the best Configuration of Each Pruning Strategy

of disabling helpful macro-action pruning is greatest in 6 of the 8 evaluation domains, where the planner solves far fewer problems, and takes much longer to solve mutually solved problems. The impact is lower in the remaining 2 domains, Airport and FreeCell, as here many of the problems are solved when EHC aborts and best-first search is used. In best-first search, neither helpful action nor helpful macro-action pruning is applied.

Using the Wilcoxon Signed Rank Test, we can show that the 'Keep All' configuration achieves a significant reduction in time taken to find solutions to mutually solved problems than the 'No Caching' configuration. Further, 'No Caching' solves problems significantly more quickly than 'No Macro-Actions'. These tests are carried out to a confidence level of 97.5% to show that the total ordering—caching macro-actions is better than no caching is better than no macro-actions—holds to a confidence level of 95%.

5.2 Macro-Action Ordering

In this section we evaluate the impact on planner performance of the ordering of macro-actions with respect to the unit actions. To do this, the 'Keep All' configuration of the planner (keeping all macro-actions found) was modified to give several planner configurations ordering differing numbers of macro-actions before the unit-actions. We also made similar modifications to the 'No Caching' configuration (in which macro-actions are discarded after solving each problem). The results obtained from the extremes of these configurations are shown in Figure 2.

It can be seen that of the 'Keep All' configurations, considering all macro-actions after the unit actions ('Keep All, All After') is the most successful strategy, with more problems being solved overall. The worst performance is observed when all the macro-actions are ordered before the unit-actions ('Keep All, All Before'). The performance is poor in this case as many more states are evaluated during search due to more macro-actions being considered. The helpful macro-action pruning reduces the impact of this to a certain extent, allowing some problems to be solved successfully, but the planner clearly gains the most when all macro-actions are ordered after the unit actions, thus con-
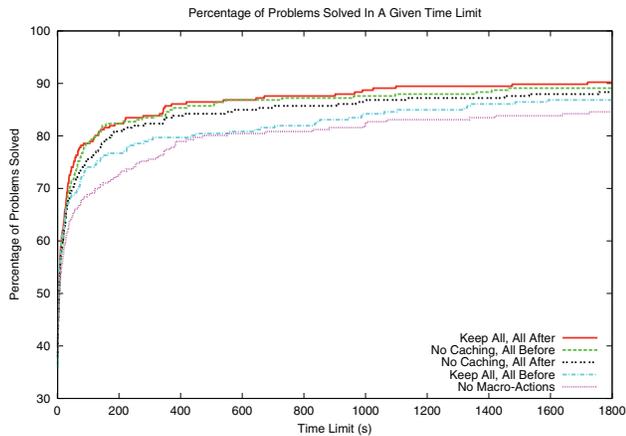
Figure 2: Percentage of Problems Solved in a Given Time Limit Considering Different Numbers of Macro-Actions Before Unit Actions

sidering them only when plateaux are reached.

In the 'No Caching' configurations, the results show a performance gain when ordering macro-actions before unit actions. As macro-actions are not kept between solving problems, far fewer macro-actions must be considered during search as only those generated on the current problem are available. In the Briefcase domain, ordering macro-actions before unit actions leads to improved performance. Only two macro-actions are generated in this domain, and when a macro-action is applicable, applying it is the best course of action. Similar performance gains can be observed in the Satellite domain, and to a lesser extent, in the Depots domain. The performance gains arise in domains where small collections of macro-actions are generated and in which when applicable, often suggest a good direction in which to proceed. Unfortunately, when scaled to a large collection of macro-actions, such as in the 'Keep All' configurations, the same gains are not seen (other than in the Briefcase domain where only two distinct macro-actions are ever generated). As such, overall, the consideration of macro-actions after the unit actions allows the search behaviour of the planner to scale better to reason with large collections of macro-actions. This allows large collections of macro-actions to be used, and outweighs any gains made through considering macro-actions before unit actions.

5.3 Library Management Techniques

Usage Count
Two techniques for macro-action library management based on the usage counts for each macro-action were presented in Section 3.1. These were the *survival of the fittest pruning strategy*, maintaining only the most used macro-actions in the library, and the *roulette selection strategy* selecting macro-actions probabilistically based on their usage counts. The best result obtained from managing libraries using usage counts is shown in Figure 1, labelled 'Top 10'. Due to space constraints other results are omitted; tests were, however, ran with a range of parameters for all strategies.

In the survival of the fittest pruning strategy, configurations were tested in which the top 10, 5 and 2 macro-actions

were kept; these solved 92.9%, 90.6% and 89.5% of the problems respectively. The most successful configuration was maintaining the top 10 macro-actions, indeed this was the most successful of all the library management strategies, as shown in Figure 1 (labelled 'Top 10'). Both the top 10 and top 5 configurations solve more problems than the 'Keep All' configuration. We have shown in section 5.1 that our powerful helpful macro-actions pruning technique allow the planner to function effectively with a large macro-action library. The results from this experiment confirm that library management can indeed further reduce the time taken to solve problems. Using the Wilcoxon Signed Rank Test we can confirm the top 10 configuration achieves a significant reduction in time taken over the 'Keep All' configurations.

The effectiveness of the search-time helpful macro-action pruning is sufficient to allow the planner to maintain much larger libraries than one might naturally expect. Repeating the survival of the fittest pruning strategy tests with helpful macro-action pruning disabled presents a different picture of the results. Without helpful macro-action pruning, the 'No Macro-Actions' configuration solves the most problems, 82.7% of the collection. Performance degrades as the number of macro-actions kept increases: the 'No Caching' configuration solves 78.9% of the problems; keeping only the top 2 is the most successful caching configuration, solving 77.4%; and 'Keep All' solves only 73.7%.

The roulette selection strategy is not as successful as survival of the fittest. None of the configurations successfully solve more problems than the 'Keep All' configuration, in which no library management is used. Varying the parameters shows that reducing n (i.e. selecting fewer macro-actions from the library) degrades performance, as the chance of selecting the better macro-actions is reduced.

Additional experiments were performed to address the issue of whether the number of times a macro-action is selected is a good indicator of its potential usefulness. A configuration of the planner was developed that kept 10 macro-actions, chosen at random from the library, rather than the top 10 in terms of usage counts. The comparison of this configuration to the top 10 configuration negates the possibility that the benefits could simply be arising from differing macro-action library sizes. The 10-at-random configuration was able to solve only 87.6% of the evaluation problems, compared to the 92.9% solved by the top 10 configuration. Further, it is also not as successful as the 'No Caching' configuration, which solves 88.3% of the problems. This confirms that it is indeed keeping macro-actions on the basis of usage counts that is providing performance benefits, not simply trimming the library to the smaller size.

Time Since Use
Pruning macro-actions based on the number of problems solved since they last appeared in a solution plan forms the basis of the *time out* pruning strategy. Several time out values were considered, removing macro-actions from the library if they have not been used for 1, 2, 5 or 10 problems. The most successful of these configurations was the macro-actions were removed if they were not used for 1 problem, that is they are guaranteed to be kept for use in solving the next problem, but if they are not then used they

are removed. This gives rise to performance slightly better than the 'Keep All' configuration. This is the only configuration of this pruning strategy that performs better overall than 'Keep All'; specifically, it solves one more problem. This pruning strategy does not demonstrate great success for a number of reasons. The first is that the number of macro-actions kept in the library is not significantly reduced, so the benefits of having a small number of useful macro-actions to consider are not realised. Further, some of the better macro-actions are pruned from the library, so in general the performance is worse than if all macro-actions are kept.

In some domains, such as Philosophers, there is an oscillating pattern of when macro-actions are useful. In Philosophers some macro-actions are useful on every other problem, so a time-out of 1 problem can cause useful macro-actions to be lost. Other domains, although not following regular patterns, still exhibit the problem of losing useful macro-actions from the library if they have not been used in solving the past few problems. Once a macro-action has been lost from the library, if it is found again when solving a later problem and added to the library, its usage count will be zero. As macro-actions are always considered in usage-count order, the likelihood of it being used is therefore reduced, and it is likely that it will once again be removed from the library due to disuse.

Instantiation Count

The *instantiation versus use* pruning strategy makes use of both the instantiation and usage counts of macro-actions, removing a macro-action from the library if $usage count / instantiation count$ is less than a threshold, θ. Experiments were performed varying the value of θ; the optimum value found was $\theta = 0.1$, leading to 89.5% of problems being solved. The performance of this configuration is similar to that of the 'Keep All' configuration, the results for which are shown in Figure 1. Instantiation versus use pruning offers similar performance to keeping all macro-actions. Its advantage, however, is that a much smaller macro-action library is maintained. Potentially, in a planner where helpful macro-action pruning was not used, this could give a performance improvement over keeping all macro-actions.

In some domains, in particular those with directed search spaces, some of the instantiation versus use configurations allow the planner to solve more problems than have been solved by previous strategies. Overall, across all domains, this is a successful library pruning strategy generating small, high performing libraries. However, due to the aggressive helpful macro-action pruning employed, these smaller library sizes do not give rise to a performance improvement.

5.4 Length of Macro-Actions

It is frequently postulated that short macro-actions are the most likely to be useful and applicable; whilst long macro-actions are either rarely applicable or produce too many possible instantiations. To test this, configurations of the planner were tested in which the macro-action lengths were restricted to being at most 2, 3, 4 or 5. Of these, restricting the length to 2 offers the best performance; solving 89.1% of the problems. This performance is worse than that of the 'Keep All' configuration.

Analysis of the macro-actions found reveals that it is indeed the case that macro-actions of length 2 are used more frequently than other macro-actions, occurring a total of 3156 times across solutions to the evaluation problems solvable within the imposed time and memory limits. This is in contrast to macro-actions of length 3, used a total of 698 times, and macro-actions of length between 4 and 10 inclusive, occurring fewer than 25 times each. However, there are useful macro-actions of length 11, in particular in the philosophers domain, with macro-actions of length 11 being used a total of 600 times in solving the whole problem suite. Similar long macro-actions, of length 8, are also found in the Optical Telegraph domain (not included in the evaluation suite to promote diversity of domains). As such, long macro-actions which are useful in some domains, need to be preserved to maintain overall performance.

Macro-actions of length 3 are often used in some domains; for example in Driverlog, macro-actions of length 3 are used as often as those of length 2. Examining the usage counts of each macro-action generated in the Depots domain, macro-actions of length 3 occur less often in solution plans than macro-actions of length 2. However, an arbitrary macro-action of length 3 has a greater chance of being used than one of length 2, as fewer macro-actions of length 3 are generated and have higher individual usage counts.

The usefulness of macro-actions with lengths greater than two, combined with the worse performance obtained when restricting the length of macro-actions to 2, shows that the length of macro-actions is not necessarily a good indicator of whether they should be discarded. In our tests, macro-actions of length greater than 2 accounted for 30% of the macro-actions used in solution plans, making a valuable contribution to problem solving. Other features of macro-actions have been shown to be more successful for pruning macro-action libraries, without losing valuable long macro-actions that can potentially save a great deal of search effort.

5.5 Solution Quality

It is often noted that the use of macro-actions in planning has the potential to reduce solution quality. Our final set of results show that, in fact, far from decreasing solution quality, our macro-actions actually have a positive effect on the makespan of problems. We consider three configurations of the planner for this experiment: the standard 'No Macro-Actions' configuration; 'No Caching', generating macro-actions only on a per-problem basis; and 'Keep All', which caches all macro-actions generated.

Table 1 shows the mean makespan improvement of the 'No Caching' and 'Keep All' configurations over the 'No Macro-Actions' configuration. When macro-actions are included in the plan their contribution to the makespan is their total length. Overall, both of these generate shorter solution plans than the 'No Macro-Actions' configuration. Using macro-actions is particularly successful in the Briefcase domain, where the heuristic otherwise guides the planner to add unnecessary actions to the plan. The macro-actions offer alternative guidance, allowing the planner to apply a macro-action to move the briefcase to a location and then load an object into it. Without this macro-action, the planner reaches

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 1: Mean Makespan Improvement Achieved Over The 'No Macro-Actions' Configuration

a plateau and blindly explores visiting several states until an appropriate one is found.

In addition to the large benefits in Briefcase, macro-actions allow the planner to produce solutions that are at least as short as those found without macro-actions in 5 out of the remaining 7 evaluation domains. There are three reasons for the benefits being observed. The first is the concurrency present in the macro-actions: Marvin introduces concurrency into macro-action schemata when they are generated, placing non-mutex adjacent actions in parallel. Second, macro-actions allow more problems to be solved by EHC, rather than resorting to best-first search. As Marvin reasons about concurrency in EHC but not in best-first search (due to the expense of doing so) shorter plans are usually generated when EHC is successful. Finally, macro-actions found on smaller problems often allow shorter plateau exit routes to be found on larger problems. When plateaux are escaped in smaller problems, the length of the plateau-escaping action sequence chosen is likely to be closer to optimal than in large problems; where the likelihood of inserting irrelevant actions is greater.

The Driverlog and Pipes-NoTankage domains are the only two domains in which the makespan is increased by the use of macro-actions. In the Driverlog domain, in general, best-first search (used when EHC fails) finds shorter plans than EHC; but macro-actions allow EHC to solve more of the problems, thereby indirectly increasing the makespans of solutions found. In the Pipes-NoTankage domain, the plateaux can have very long exit depths and some macro-actions generated on long plateaux also successfully escape shorter plateaux. The increase in makespans of plans is, however, only slight: around two time steps.

Overall, despite slight makespan increases in Pipes-NoTankage and Driverlog, the Wilcoxon Ranked Sign Test shows with 97.5% confidence a significant makespan reduction when 'No Caching' is compared to 'No Macro-Actions' and when 'Keep All' is compared to 'No Caching'. As such, a total ordering holds on these with 95% confidence.

6 Conclusions

We have presented an online learning approach for the management and generation of macro-actions in planning. Our approach does not require the solving of training problems, with and without macro-actions, as an offline learing step prior to search. We have shown that using macro-actions generated online allows a significant performance improvement over not using macro-actions, in terms of both time

taken to solve problems and solution quality. Also, we have shown that storing these macro-actions in a library for use on future problems can offer a further significant improvement.

We have presented powerful search time pruning techniques, in the form of helpful macro-action pruning and macro-action ordering, that allow large libraries of macro-actions to be maintained without the potential branching factor increase outweighing the benefits. Further, we have shown that pruning the macro-action library offers a performance improvement over keeping all macro-actions through the use of the *survival of the fittest* pruning strategy, keeping the 10 most-used macro-actions. We also shown that *instantiation versus use* pruning allows similar performance to keeping all macro-actions whilst maintaining smaller macro-action libraries; this will be important in systems where the search time pruning we used is not applicable. Our approach to library management and pruning is general, and can be combined with any online macro-action learning technique in which it is reasonable to expect that the macro-actions learnt are likely to reduce solution time if used.

7 Future work

We hope to extend our investigations in the future to analyse the sensitivity of the techniques to the order in which the planner is presented with the problems. We would also like to investigate the applicability of plateau-escaping macro-actions in other planning paradigms, such as the local search framework of LPG (Gerevini & Serina 2002).

References

Blum, A., and Furst, M. 1995. Fast Planning through Planning Graph Analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Inteligence (IJCAI-95)*, 1636–1642.

Botea, A.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research* 24:581–621.

Coles, A., and Smith, A. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research* 28:119–156.

Gerevini, A., and Serina, I. 2002. LPG: A Planner based on Local Search for Planning Graphs. In *Proceedings of the Sixth International Conference of Artificial Intelligence Planning and Scheduling (AIPS-02)*, 13–22.

Hoffmann, J., and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research* 24:519–579.

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

Long, D., and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Newton, M.; Levine, J.; and Fox, M. 2005. Genetically evolved macro-actions in A.I. planning problems. In Tuson, A., ed., *Proceedings of the 24th UK Planning and Scheduling SIG*.

Smith, A. 2007. *Online Generation and use of Macro-Actions in Forward-Chaining Planning*. Ph.D. Dissertation, University of Strathclyde.