Simulating the use of Macro-Actions Through Action Reordering

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

Abstract

The use of macro-actions in planning introduces a trade-off. Macro-actions can offer search guidance by suggesting sequences of actions; but can potentially make search more expensive by increasing the branching factor. In this paper we present a technique for simulating the use of macro-actions by altering the order in which actions are considered for application during enforced hill-climbing search. Actions are ordered based on the number of times they have occurred, in past solution plans, following the last action added to the plan. We demonstrate that the action-reordering technique used can offer improved search performance without the negative performance impacts often observed when using macro-actions.

Introduction

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.

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: the increased branching factor at each action choice point in the search. In addition to considering the applicable instantiations of the actions in the domain, instantiations of 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 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 instantiations 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. Thus, if a macro-action of length k is used the complexity will be reduced to O($(b+n)^{l-k}$).

We present a technique to simulate the use of macro-actions, without increasing the branching factor, by changing the order in which actions are considered for application during search. This avoids the potential problems typically associated with using macro-actions. Actions are ordered according to past records of action sequences occurring in solution plans. Our technique is implemented in the planner Marvin (Coles & Smith 2006) that uses an FF-style planning framework (Hoffmann & Nebel 2001): enforced hill-climbing search with best-first, rather than breadth-first, search on plateaux.

Related Work

Macro-actions were first conceived in the early stages of planning research when solution plans were used as macro-actions for solving future problems. The use of macro-actions was developed by many researchers, some concentrating on pruning techniques and selecting which macro-actions to use (Minton 1985; Iba 1989; McCluskey 1987) and others on techniques for extracting macro-actions from the domain itself (Dawson & Siklossy 1977; McCluskey & Porteous 1997). Research into the use of macro-actions has enjoyed a recent surge in popularity, with two planners in the Fourth International Planning Competition making use of macro-actions. Macro-FF (Botea, Muller, & Schaeffer 2004) uses an offline learning process to learn macro-actions for each domain. In the most successful version, macro-actions are lifted from solution plans and then filtered using an offline training process that involves solving small problem instances. Marvin (Coles & Smith 2006) uses an online learning technique to generate macro-actions, based on plateaux in the search landscape. The macro-actions are generated online and can also be stored for use in solving later problems (Coles & Smith 2005). Work on evolving macro-actions (M.A.H. Newton 2005) has also been done. As with Macro-FF, this process is another that uses offline learning, using a genetic algorithm to generate and filter macro-actions.

Action-reordering techniques have been exploited to achieve two different objectives: to increase the efficiency of the planning process; and to allow the introduction of concurrency into solution plans. Work on symmetry in planning (Fox, Long, & Porteous 2005) has been used to suggest an action ordering to a planner. The helpful actions, as generated by FF, are ordered to consider first those actions that are 'almost symmetric' with actions applied at some previous time point in the plan. Almost symmetry attempts to capture similarities between objects in the domain that are not functionally symmetrical but are in very similar situations. Such objects are likely to require similar action applications in order to reach their desired goal state; hence, actions are reordered to consider actions that have already been applied to an almost symmetric object before other actions.

CRIKEY (Halsey 2004) 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 planner, 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. 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. The action reordering allows for greater parallelisation of the final plan and improved makespans of the plans generated by CRIKEY.

The action reordering presented here is also related to work on control rules (Bacchus & Kabanza 2000; Martin & Geffner 2004). Control rules suggest to the planner the next action to apply, usually based on propositions that are true in the current state. The action reordering in this work also suggests the next action to apply, but this suggestion is based on the previous action in the plan.

Action Reordering

Macro-actions that are used frequently represent sequences of actions that follow each other often. An action ordering strategy based on the number of past occurrences of a given action following another action can be used to implicitly suggest macro-actions to the planner, without actually adding the macro-actions 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 additional actions.

For example, the macro-action `pickup-move-drop` in the Gripper domain might be extracted by a macro-action-generating planner due to its frequent occurrence, it being a plateau in the search space, or due to it improving search performance. It is also possible, however, to simply observe that so far during search to solve a problem (and indeed in search to solve previous problems) that every time a `move` action has been applied, a `drop` action was applied afterwards; and thus, it would be sensible to consider `drop` actions after `move` actions in future search. Similarly, a `move` action may have followed a `pickup` action 50% of the time; the other 50% of the time a `pickup` action is likely to have been followed by another `pickup` action (one `pickup` for each gripper). Thus, it is sensible to consider applying `move` and `pickup` actions after following the application of `pickup` actions, before considering other actions.

The basis of the action reordering strategy is to order the list of potential action choices by an estimate of the likelihood that it is correct to order them after the last chosen action in the plan, based on the frequency with which the two actions have occurred one after the other. The frequencies for each action following each other action are initialised to zero. As planning progresses, each time a given action follows another its 'following frequency' is increased. When a node is to be expanded in search, actions leading to successors are ordered according to this frequency metric, and their corresponding successor states are expanded in the given order. In EHC, as the first action which leads to a state with a strictly better heuristic value is chosen, if the first ordered action leads to such a state, a great deal of search effort will be saved.

The implicit macro-actions suggested are a slightly weaker form of macro-actions than those in the traditional sense. Although a sequence of actions is suggested to the planner, which parameters should be shared between adjacent actions is not explicitly enumerated, as would be the case when using a macro-action. Maintaining the data of shared parameters for action ordering would become very expensive; the ordering is therefore based only on the action types and not on any shared parameters. For example, in the gripper domain, if the learnt action data suggests that `drop` is the most likely action to follow `move` then, if a move action has just been added to the plan, the actions will be ordered such that all the `drop` actions occur before the others, irrespective of their parameter bindings. Macro-actions, on the other hand, would effectively order a specific instantiation of a `drop` action first.

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 is comprised of interleaved actions to perform tasks in several distinct locations. For one package, the plan fragment may be: `(load package1 truck1 location1) (drive truck1 location1 location2) (unload package1 truck1 location2)`; and similarly, for another package in another location: `(load package2 truck2 location3) (drive truck2 location3 location4) (unload package2 truck2 location4)`. This could be further interleaved with the end of another earlier sequence concluding with `(unload package3 truck3 location5)`. These plan fragments can be interleaved arbitrarily to create the overall plan; an example is given in Figure 1.

Updating the action-following data to reflect the orderings in the interleaved plan could be misleading: for example, it

```
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 1: Example Plan Segment for the Driverlog Domain

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 in general it is not the case 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 ordering data between adjacent pairs of actions in the plan, regardless of threads of execution, is referred to as U1 in the evaluation. To attempt to address issues regarding thread independence in plans, further reordering strategies have been developed and are described in the following sections.

Update if the Previous Action Shares A Parameter

In first of these more sophisticated strategies, U2, action frequency data is only updated between pairs of actions in the plan which share a common parameter. Actions sharing a parameter are often reliant on some common object in the world, and so can be considered linked as part of a sequence of execution. Updating the data between only pairs of 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 between adjacent pairs of actions if they share a common parameter, information is potentially lost about orderings between related but non-adjacent action pairs.. 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 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 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. It can be seen that the fact that `unload` follows `drive` would not be recorded anywhere 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 a

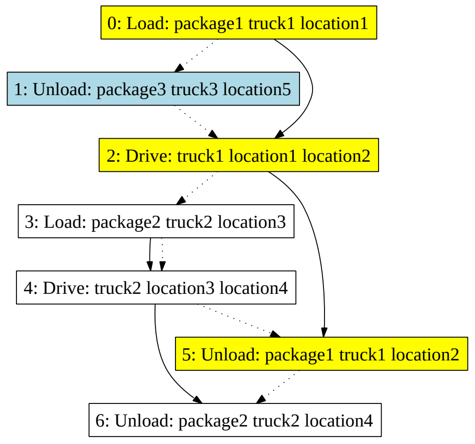
0: Load: package1 truck1 location1
1: Unload: package3 truck3 location5
2: Drive: truck1 location1 location2
3: Load: package2 truck2 location3
4: Drive: truck2 location3 location4
5: Unload: package1 truck1 location2
6: Unload: package2 truck2 location4

Figure 2: Threads of Execution in Driverlog Plan Segment. Dotted lines indicate dependencies implied by the plan structure; solid lines indicate true dependencies.

single action A before an action B, all the actions immediately before A that are pairwise non-mutex with each other and A are considered. The pairwise non-mutex actions represent a collection of actions which could have been ordered arbitrarily in the plan: there are no ordering constraints between them. An action added after these could follow on from any one of these actions; not just from the action that happens to be ordered last. To identify these potential predecessors when ordering the actions to be considered, the algorithm reverses through the plan built so far collecting actions until one is found that is mutex with any of the actions that follow it. At this point, regression stops and the action mutex with one or more of those following it is excluded from the set of potential predecessors, as it necessarily precedes another action in the predecessor set.

Consider the addition of the final two `unload` actions to the plan in the example in Figure 1; for clarity Figure shows the separate threads of execution in this plan. Upon adding the action `(unload package1 truck1 location2)` at step 5 to the plan, the algorithm regresses through the plan to find possible predecessors to update the action-following data. 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. Search for a solution plan then continues and the action `(unload package2 truck2 location4)` is selected for addition at time step 6. Regressing through the plan before the newly added action, 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 precede the action at time step 6 in the same thread of execution; that is, any action in the 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; thus, the number of times that an unload action has followed a drive action is incremented accordingly.

Cached Probabilistic-Observation-Based Action Reordering

Having now described how action-following data can be recorded, online, during planning, the strategy can be extended to cache the table of action-following data between runs of the planner in a given domain. This data can then be used to suggest orders in which to consider actions when solving subsequent problems. Unlike the libraries of cached macro-actions, the size of these tables does not increase as more problems are solved: the information stored in the table simply becomes a more-accurate reflection of the action orderings observed when planning. There is, therefore, no need to employ any library management or pruning strategies in order to maintain the library.

Results

The techniques discussed have been implemented in the planner Marvin (Coles & Smith 2006). The plateau-escaping macro-action generation techniques and concurrent planning framework are disabled throughout these experiments; resulting behaviour analogous to FF, other than using best-first search, instead of breadth-first search, on plateaux. Tests have been performed across a wide range of domains with very different properties. All experiments have been performed on a Linux machine with 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. Where available the ADL version of each domain is used; if not, the typed STRIPS version of the domain is used. All problems are taken from the recent competition benchmark suites in IPC 3, 4 and 5; except for Briefcase (which did not feature in these competitions) in which 20 randomly generated problems are used with the number of objects being increased from 50 to 240 (in increments of 10). These competition problems are the standard benchmarks used in the comparison of planners. Some of the problems were designed to model matters of academic interest; others model real world problems, for example, the Airport domain is concerned with managing airport ground traffic in Munich airport.

Using Reordering Data on a Per-Problem Basis

The results shown in table 2 show the improvement in time taken to solve problems using each of the update strategies with action reordering, relative to the control version of the planner. Precisely, the data presented in the table is $\sum_{i=0..n} (T(C_i) - T(V_i))$, where n is the number of problems in the domain and T(C_i) and T(V_i) represent the time taken by the control and version of the planner to be considered, respectively. Only problems solved by both the individual configuration and the control are included in calculating these results: the figures in brackets indicate the number of mutually solved problems.

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
TPP	30	30	30	30
Totals	**256**	**259**	**259**	**250**

Table 1: Coverage Across Evaluation Domains Using Action Reordering with Different Update Strategies

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 1). 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. In these situations 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. In doing so, the planner is able to successfully solve these problems via 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 in this domain; however, 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.

In the Philosophers domain the performance of the planner is consistently improved over the problems in the evaluation suite. Solving the problems up to that with 49 philosophers gives the configurations making use of action reordering a mean improvement in time taken of just over 12 seconds. The time taken by each of the configurations grows according to a regular function, and the time taken by the reordering version is consistently lower.

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 find-

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)*
TPP	11.61 *(30)*	11.6 *(30)*	-1.61 *(30)*
Totals	**6.82** ***(256)***	**6.96** ***(256)***	**1.50** ***(256)***

Table 2: Mean of time taken by control version minus time taken using each update strategy with action reordering. Results are calculated on mutually solved problems.

ing a strictly better state at every choice point. The planner does not, however, make the same gains that are observed when using macro-actions as action reordering does not offer a major benefit during best-first search on plateaux. The search will have to explore all states 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 2 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 some 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 limit.

In the Satellite domain the U1 and U2 configurations 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 one 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 predecessor actions, and suggests in the future using a `turn-to` action to follow another `turn-to` action. The other learning strategies learn from 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 made significantly worse: the planner is only learning from the mistakes it would usually make, and the mistakes are still made in search whether they are reinforced or not. The results in the TPP domain show a similar pattern to those in the Satellite domain with reordering showing a positive impact in the U1 and U2 strategies, and a slight negative impact when using the U3 strategy.

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: no regular pattern emerges showing action reordering to produce generally better, or worse, behaviour. Most problems in this domain 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 configuration 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.

Caching Reordering Data

In this section the configurations of the planner presented differ from the previous section only in that the action-following data is cached between problems. 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.

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

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
TPP	30	30	30	30
Totals	**256**	**263**	**261**	**260**

Table 3: Coverage Across Evaluation Domains Using Action Reordering with each of the Update Strategies, Caching Reordering Data

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 action-following data becomes meaningful.

Table 3 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.

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. The performance of the U3 version 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, leaving uncollected items at locations which must later be collected.

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)*
TPP	9.83 *(30)*	9.82 *(30)*	13.04 *(30)*
Totals	**8.55** ***(255)***	**10.46** ***(254)***	**-1.31** ***(252)***

Table 4: 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.

All versions of the planner have solved all the problems in the Philosophers and Satellite domains; therefore, the data in Figures 2 and 4 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 consistent improvements made in this domain can be seen in Figure 3. 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.

The results in the TPP domain show that this is another domain in which the action reordering data can improve per-

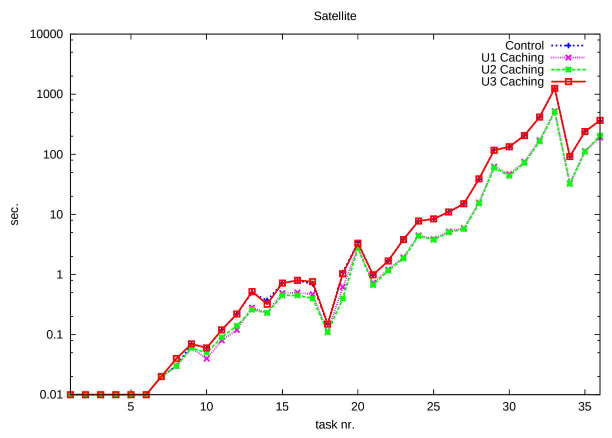

Figure 3: Using Action Reordering in the Satellite Domain

formance on almost all problems. The results for U1 and U2 caching show that the caching versions using this update strategy do not perform as well as those learning the reordering data on a per-problem basis. The U3 version, however, shows better performance when caching the reordering data allowing for a positive impact on performance, rather than the slight negative impact seen when not caching data.

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 configurations 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 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 the FreeCell domain leads to the U1 and U2 versions of the planner being able to solve all problems. 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 configurations of the planner are shorter in all but one of the problems. 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.

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)*
TPP	12.17 *(30)*	12.17 *(30)*	14.53 *(30)*
Totals	**2.50** ***(255)***	**1.97** ***(254)***	**2.94** ***(252)***

Table 5: 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.

Table 6 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 $\sqrt{0.95}$).

Conclusions

We have presented a technique for simulating the use of macro-actions through action reordering and shown that it can enhance planner performance across a diverse range of domains. In some domains the performance improvement is slight; in others it is more significant. Unlike when using macro-actions, however, the performance is rarely significantly degraded by reordering the actions as the branching factor is not increased.

In these experiments the different probabilistic update strategies all exhibit very similar performance except in the Satellite domain. The action reordering techniques have

Update Strategy	**Control vs No Caching**	**Caching vs No Caching**	**Total Ordering Significant at 95% ?**
U1: p ≤	$1.557 * 10^{-05}$	$4.12 * 10^{-07}$	Yes
sig?	Yes	Yes	
Better	No Caching	Caching	
U2: p ≤	$1.187 * 10^{-05}$	$3.081 * 10^{-07}$	Yes
sig?	Yes	Yes	
Better	No Caching	Caching	
U3: p ≤	0.1203	0.0004285	No
sig?	No	Yes	
Better	Control	Caching	

Table 6: 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 $\geq$ 0.975; Better is the best performing of the two configurations being compared.

very little impact on the makespan of solution plans. The makespan data across all domains, shown in Figure 5, shows pleasingly that the action reordering strategy is offering a slight improvement in makespan and not leading to longer plans, a problem often associated with macro-action use.

When learning action reordering data on a per-problem basis performance improvements can be seen in some domains; the guidance offered by action reordering is, however, improved when the data is stored for use on future problems. 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.

Future Work

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. Another similar extension would be to investigate extending the technique to reason about shared parameters to simulate macro-actions more strongly.

References

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

Botea, A.; Muller, M.; and Schaeffer, J. 2004. Using component abstraction for automatic generation of macro-actions. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, 181–190.

Coles, A. I., and Smith, A. J. 2005. On the inference and management of macro-actions in forward-chaining planning. In Tuson, A., ed., *Proceedings of the 24th UK Planning and Scheduling SIG*.

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

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

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

Halsey, K. 2004. *CRIKEY! It's Co-ordination in Temporal Planning: Minimising Essential Planner–Scheduler Communication in Temporal Planning*. Ph.D. Dissertation, University of Durham.

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

Iba, G. A. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3:285–317.

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

Martin, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20(1):9–19.

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65.

McCluskey, T. L. 1987. Combining weak learning heuristics in general problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI '87)*.

Minton, S. 1985. Selectively generalizing plans for problem-solving. In *Proceedings of the ninth International Joint Conference on Artificial Intelligence, IJCAI '85)*.