# On The Inference and Management of Macro-Actions in Forward-Chaining Planning

**Andrew Coles** 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

In this paper we discuss techniques for online generation of macro-actions as part of the planning process and demonstrate their use in a forward-chaining search planning framework. The macro-actions learnt are specifically created at places in the search space where the heuristic is not informative. We present results to show that using macro-actions generated during planning can improve planning performance.

## 1 Introduction

Previous work on using macro-actions in planning has used macro-actions generated in advance of the planning process, either through additional off-line learning steps—such as those used by Macro-FF [Botea *et al.*, 2004]—or through hand-coding—such as the 'tasks' used in hierarchical planning [Nau *et al.*, 2003]. We present an approach that differs from previous work in that it is able to infer macro-actions on-line, during the process of searching for a plan, as well as being able to handle macros cached from previous planning problems. The macro-actions inferred specifically aim to provide search guidance at the points during search where the heuristic used is unable to provide useful information.

The underlying planning framework in which the macro-action inference (and use) strategy described in this paper has been developed is forward-chaining heuristic-search. The search algorithm used is enforced hill-climbing (EHC) guided by the relaxed planning graph (RPG) heuristic, as in FF [Hoffmann and Nebel, 2001]. EHC is a gradient-descent local-search algorithm, using exhaustive search to escape plateaux. The macro-action inference strategy is an important feature of Marvin [Coles and Smith, 2004], a planner that participated in the Fourth International Planning Competition (IPC4). Marvin extends the search used by FF to allow exploitation of concurrency, and to provide support for the inference and use of macro-actions during planning; these modifications are described in this paper.

## 2 Terminology

Marvin can handle PDDL [McDermott, 2000; Fox and Long, 2003] planning problems specified using ADL and derived predicates [Edelkamp and Hoffmann, 2004]. It cannot reason about temporal planning or numeric expressions. Thus, a *planning problem* can be defined as a tuple $<$I, G, A$>$, where I and G are conjuncts of literals representing the initial and goal states, and A is the set of lifted action schemata. A *ground action* is one whose parameters have been bound to specific entities; a *lifted* action has at least one un-bound parameter. A solution *plan* to such a planning problem, as presented by Marvin, consists a partially-ordered series of time-stamped, ground, primitive actions.

During search, both *primitive actions* and *macro-actions* are available. Primitive actions are those defined by the action schemata in the planning domain; they can be defined by a tuple $<$P, E$>$, where P is an arbitrary well-formed propositional logic formula detailing the preconditions, and E is a list of effects the action has (add, delete, or conditional) upon being applied. Macro-actions can be defined by a tuple $<$P, A$>$, where P is a well-formed propositional logic formula detailing the preconditions necessary for the application of the macro-action, and A is a plan consisting of lifted actions. As the primitive actions may have conditional effects, the external effects of the macro-actions are not known until the macro-actions are ground.

Marvin makes use of symmetry between objects in the states that arise during planning. The *equivalence function* used to define the symmetries is the *functional symmetry* defined in [Fox and Long, 1999]: entities are symmetric if they are of the same type, are described by the same predicates in the current and goal state, and do not appear (as constants) in the preconditions or effects of any of the operator schemata. A *symmetry group* is a group containing all mutually-symmetric objects in a given state.

## 3 Plateau Escaping Macro-Actions

Many planning problems exhibit a large amount of symmetry: whilst symmetry in the problem instance itself has been exploited [Fox and Long, 1999; Rintanen, 2003]; it is difficult to exploit symmetry in solution plans. The difficulty arises because the final solution plan is not known until the planning process is complete: as such, during the planning process, it is not obvious where the symmetry in the solution plan will be. Symmetry in plans is characterised by recurrent action sequences. Such sequences represent some underlying symmetry between objects in the domain: they often occur because many symmetric objects require the same sequence of actions to transform them from their initial state into the final goal state. If potential recurrent action sequences can be identified during search for a solution plan and encapsulated to form macro-actions a potential reduction in planning time can be made—particularly if these action sequences are difficult to find. Also, if a reduced instance of the original problem is derived, and solved prior to the original problem instance [Coles and Smith, 2004], the recurrent action sequences could be identified on a smaller problem, providing further gains.

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

On inspection of the steps required to escape plateaux in a given domain it is often the case that the same sequence of actions is used to escape many plateaux, but with different parameter bindings. For example, in the depots domain, from the Third International Planning Competition (IPC3), the pattern lift-load is frequently used with different groundings of the actions. In the philosophers problem, part of the Promela domain from IPC4, a complex action sequence is used many times, once for each pair of philosophers.

As exhaustive search is required to escape from a plateau, construction of plateau-escaping action sequences is computationally expensive. Since plateau-escaping sequences often have similar structure, it is clear that memoising these action sequences for later use—when plateaux are once-again encountered—can potentially reduce planning time. In Marvin the plateau-escaping action sequences are used to construct *plateau-escaping macro-actions*. Once devised, they are made available, as actions, for application on later plateaux.

## 3.1 Inferring Plateau-Escaping Macro-Actions

When the start of a plateau is detected—that is, when no successor state with a strictly-better heuristic value can be found—best-first search commences from the current state. During best-first search, each successor state stores the actions that have been applied to reach it since the start of the plateau: when a strictly-better state is eventually found, this list of actions is the plan segment that forms the basis of the plateau-escaping macro-action.

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

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

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

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

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

## 3.2 The Derivation and Use of Plateau-Escaping Macro-Actions during Best-First Search

Marvin employs a modified best-first search strategy to find a solution plan if EHC fails. The modification is that when a successor $S'$ to a given search state $S$ is found with a strictly better heuristic value, the evaluation of the other successors of $S$ is postponed—$S$ is re-inserted into the search queue—and search proceeds from $S'$ (in conventional best-first search all successors of $S$ would be evaluated before $S'$ is expanded).

If all the single-action-step successors of a given state have been evaluated, and a better state has not yet been found and expanded, then the best-first search parallel to an EHC plateau has been reached: an area where all the successor states have a heuristic value worse, or no better, than the parent state. In such cases, Marvin considers applying the previously-generated plateau-escaping macro-actions to the state: in doing so, a strictly-better state can potentially be found without the need to explore many single-step actions to find the same sequence.

Marvin's mechanism for deriving plateau-escaping macro-actions is extended to allow their derivation when performing best-first search. Plateau-escaping macro-actions can be derived through plan analysis: by evaluating what the heuristic value would be at each step in the plan, segments of the plan which escaped a plateau can be identified. These segments of plan initially lead to states with a heuristic value no better than that of the best state so far, and eventually lead to a strictly-better state. During best-first search this plan analysis is performed each time a state is produced with a heuristic value better than the global-best heuristic value found so far.

As plateau-escaping macro-actions can also be derived during best-first search, Marvin is not restricted to making use of solely the plateau-escaping macro-actions that could be found during EHC—an important feature in domains where EHC is unable to make any progress.

## 4 Introducing Concurrency Into Macro-Actions

During the process of creating macro-actions from a plan a partial-order is lifted in order to allow the independent

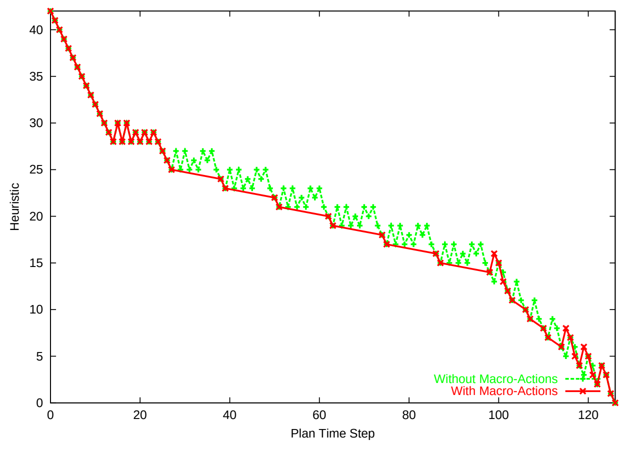

Figure 1: Heuristic Landscape over Makespan, With and Without Macro-Actions

threads to be identified. Using the partial order it is possible to introduce concurrency into the macro-actions. As long as the ordering constraints are respected, action steps the same distance from the start of a given sub-plan can occur concurrently. Reasoning about concurrency within the macro-action is only necessary when the macro-action is created: if the macro-action is used again it will already have the concurrency embedded into it.

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

## 5 Planning With Macro-Actions

Although using macro-actions during search has advantages—they can offer search guidance and allow many actions to be planned in one step—considering them during the expansion of each state increases the branching factor. Thus, if a large number of unproductive macro-actions are generated the search space will become larger, making the problem harder, not easier, to solve. Whilst many of the plateau escaping sequences are helpful in planning, some are specific to the situation in which they were derived; a situation which might not occur again in the plan. As macro-actions are learnt during the planning process—and there is no human intuition, or large test suite, to allow reusable macro-actions to be identified—care must be taken when deciding the points at which to consider their use in the planning process.

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

In Marvin, as in FF [Hoffmann and Nebel, 2001], only helpful actions are considered to be applied at each stage to reduce the branching factor (the helpful actions being those at the first time step in the relaxed plan from the current state to the goal state). In a parallel to this, only macro-actions whose first step is a helpful action are considered for application: this allows non-helpful macro-actions to be pruned, reducing the branching factor.

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

Planning with macro-actions has clear advantages, particularly if the macro-actions generated are appropriate to the domain and can be used in situations where the heuristic is unable to offer appropriate guidance. Figure 1 shows the value of the heuristic with and without macro-actions across the solution plan generated by Marvin for a problem taken from the philosophers domain (part of the Promela domain from IPC4) involving 14 philosophers. Initially, no macro-actions have been learnt so the search done by both approaches is identical. For the first 14 action choices the value of the heuristic, shown by the line in the graph, moves monotonically downwards as the planner is able to find actions to apply leading to strictly better states.

After time step 14, the heuristic value begins to oscillate, at this point the planner has reached a plateau: there is no state with a strictly better heuristic value that can be reached by the application of just one action. As this is the first plateau reached, no macro-actions have been generated so the heuristic profiles are identical for both configurations. At time step 25 a state is reached that has a better heuristic value than that at time step 14; it is at this time that the plateau-escaping macro-action will be generated, memoising a lifted version of the sequence of actions that was used to escape the plateau. A brief period of search in which a strictly better state can be found at each choice point follows before the planner again hits a plateau.

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

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

During the search to solve instances of the philosophers problem using macro-actions a large proportion of the time was spent in escaping the first plateau for the first time and the remainder of the plan was computed much more quickly. In this instance the search for the plan without using macro-actions took 596 seconds instead of 7 seconds to complete. As the instances become larger the time taken to complete each stage of exhaustive search grows dramatically and the time saved through using macro-actions becomes greater: without them it is not feasible to solve the larger instances using Marvin.

### 5.1 Reasoning About Concurrency

When using single-step actions, Marvin reasons about concurrency in forward-chaining search by considering, at the point of the expansion of each state, all actions that can be applied in parallel with the actions at the current time step as well as all those that could be applied at the next time step, where no actions are currently assigned. This approach works when planning with actions of unit length as there are only two distinct times for action application: either alongside or after the last-scheduled action. When macro-actions are to be integrated into the planning process, however, the concurrency reasoning is more complex: an action may be applicable at the same time as any of the scheduled macro-action steps, as well as after all of the steps, giving a far greater number of points at which the action could potentially be applied. It is also necessary to consider the preconditions of scheduled macro-action steps to ensure that the action to be applied does not violate any of them, leading to the production of unsound plans.

To ensure that the queued macro-action steps remain applicable, whilst still allowing as much concurrency to be exploited as is reasonably possible, the time steps over which each precondition must be maintained is calculated. Unlike temporal planning, where the invariants are maintained over the entire action duration, in the case of macro-actions these invariants may only need to be held true for part of the action duration. Further, instead of effects being restricted to occurring at the start and end of the action, there are effects that occur at each time point within it—note that, in the case of non-temporal planning, time is effectively discretised, with the single-action steps each taking one unit of time.

The effects and preconditions for scheduled macro-action steps are held in an 'event queue', similar to the one used in Sapa [Do and Kambhampati, 2001]. The event queue contains the add/delete effects, and positive/negative invariants for the future-scheduled actions. Any positive/negative preconditions belonging to action steps within a macro-action, or acting as the conditions for conditional effects within action steps, are queued to be held as positive/negative invariants from the point at which they are achieved until the point at which they are required. External preconditions are marked as invariants from the start of the event queue to the point at which they are required. An action can be applied at a given time point alongside a macro-action if it is not mutex with the cumulative invariants and effects for that point.

## 6 Caching Macro Actions

During search, Marvin generates macro-actions to use in solving each problem; in IPC4, they were generated during the planning process to solve one specific problem, and were not stored for use when solving later problem instances.

Marvin is capable of saving macro-actions to form a library of past macro-actions for use when solving other problem instances in a given domain. This is, in many ways, similar to case-based planning; the difference is that the library of past-plans (stored as macro-actions) are only sub-solutions to past problems, not whole solutions, storing only how a weakness in the heuristic was circumvented.

As in all plan libraries, a library management strategy is needed. When macro-actions are stored, they are annotated with two numbers: the total number of times they have been used so far, and how many problem files have been solved since they were last used. By storing how many problem files have been solved since given macro-action was last used it is possible to purge macro-actions from the library if they have not been used in solving any recent problem files. This is useful in domains where a phase-transition occurs as increasingly harder problems are tackled, rendering earlier-discovered macro-actions ineffective. Using the information about the number of times each macro-action has been used, the macro-actions are, when loaded, ordered in descending order of past usage: this means that popular macro-actions are evaluated first and used in preference to less-popular ones.

## 7 Results

In this section we present results that show that the performance of the planner can be greatly improved by the use of macro-actions; we also demonstrate that the overheads of using macro-actions do not necessarily adversely affect planning performance if the macro-actions are carefully managed.

Figure 2 shows the improved performance of Marvin when using macro-actions in the philosophers domain from IPC4. It can be seen that when macro-actions are cached there is a large performance improvement; the oscillation that can be seen is a result of different macro-actions being required for problems with odd numbers of philosophers than for those with even numbers of philosophers.

Inferring the plateau-escaping macro-actions on a per-instance basis brings substantial performance gains: once the periods of exhaustive search have been completed, the macro-actions inferred can be re-used and the remainder of the problem solved relatively quickly. Without the plateau-escaping macro-actions the exhaustive search step needs to

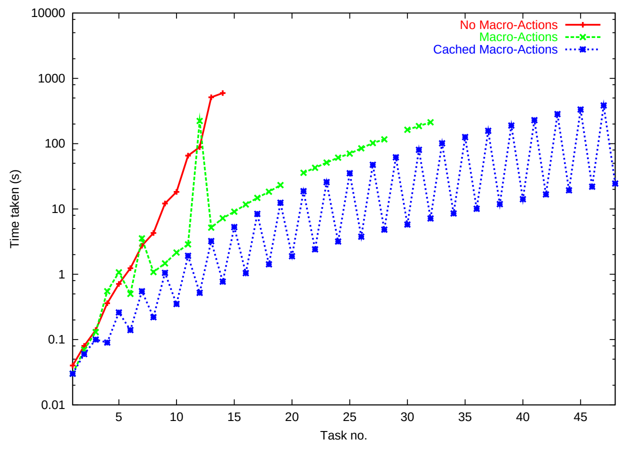

Figure 2: Coverage With and Without Macro-Actions in the Philosophers Domain

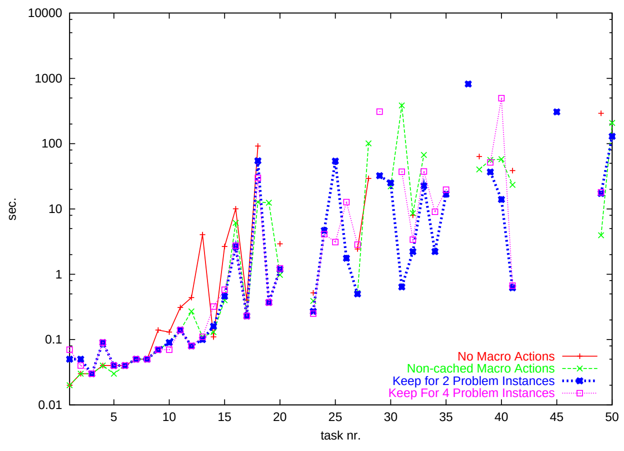

Figure 3: Coverage With and Without Macro-Action Caching in the Pipesworld Domain

be replaced many times, increasing the time taken to solve the problems.

Further performance gains are made when the macro-actions are cached between problem instances. When using only per-instance macros, Marvin is unable to solve any instances above number 33 within the alloted 30-minute time limit: this is because the first period of exhaustive search becomes too expensive to complete within the time limit, so the essential plateau-escaping macro-action is not formed. If the macro-actions are cached from earlier instances, all the problems in the set can be solved: the macro-actions necessary to bypass the plateaux are available at the start of search, having been derived on the easier problems.

The plan produced to solve each problem instance in the philosophers domain can be split into three ordered sections: an ‘activate-trans’ action per philosopher; interleavings of the first macro-action (the macro-action discovered to escape the first plateau) with the same single-step action (with differing groundings each time) trivially found to provide a strictly-better state through heuristic guidance; interleavings of the second macro-action with single-step actions, again trivially found. Once the two macro-actions have been acquired, EHC does not need to perform any plateau-escaping exhaustive search: the plateau-escaping macro-actions ‘patch’ the heuristic to allow it to always proceed to a strictly-better state. Thus, the macro-actions discovered in this domain lead to significant performance gains.

Figure 3 shows the performance of Marvin in the pipesworld-notankage-nontemporal domain from IPC4. The four configurations illustrated demonstrate the performance without macro actions; with only per-instance macro-actions; caching macro-actions which have been used in the past two problem instances; and caching macro-actions that have been used in the past four problem instances. The four configurations were able to solve 26, 33, 39 and 35 problems respectively. It can be seen that the performance of the planner when using macro-actions is sensitive to the window over which they are kept. At one extreme, with little or no caching, it can be difficult to repeatedly establish the same set of core, useful, macro-actions. At the other, non-useful macro-actions can accumulate and slow down planning performance. In this domain, caching macro-actions used over the past two problem instances gives better coverage and faster performance in most of the problems than the other two configurations. The optimal interval over which to keep macro-actions varies, depending on the domain.

A summary of the results of running Marvin on a number of different domains is shown in Figure 4. The management and use of cached macro-actions was done in the way described in section 6. It can be seen that using per-instance, and cached, macro-actions reduces planning time across a number of domains. The potential for the increase in plan length caused by using macro-actions has little effect, with the makespan in most domains being similar when the planner is run in all three configurations. In some domains the length of the plans produced is, on average, shorter when using per-instance or cached macro-actions. In such cases, the macro-actions are offering a previously found, shorter, route through the search space which the heuristic would not normally suggest in that situation.

Figure 5 shows the results of running paired t-tests to ascertain whether performance improvements are significant with 95% confidence. For all domains, excluding the philosophers domain, the performance improvement gained by using macro-actions is significant. In the philosophers domain it is not possible to prove significance to the required confidence level as the configuration not using macro-actions is unable to solve a sufficient number of problems within the time limit. It can, however be seen from figure 2 that the performance, and in particular coverage, of the planner are improved when using macro-actions, with further improvements when the macro-actions are cached.

## 8 Conclusions

We have described an approach to deriving and using macro-actions in a forward-chaining planning framework. The macro-actions are derived on-line, on a per-instance basis, and can also be cached for later use. To evaluate the effectiveness of the macro-actions formed, the approach described was implemented in a planner, Marvin.

Empirical evaluation shows that generating macro-actions on a per-instance basis is able to reduce the time taken to find solution plans in a number of planning domains; further gains can be made by caching the macros between problem instances, pruning those which do not appear to be useful.

| **Domain** | **Problems Solved** | | |
|---|---|---|---|
| | *Coverage* | *Mean Improvement* | |
| | *(%)* | *Time (s)* | *Makespan* |
| airport NMA | 80 | 0 | 0 |
| airport MA | 84 | 4.99 | 9.00 |
| airport C | 94 | 3.96 | 2.84 |
| depots NMA | 68 | 0 | 0 |
| depots MA | 73 | 85.58 | -7.00 |
| depots C | 73 | 182.90 | -3.2 |
| philosophers NMA | 29 | 0 | 0 |
| philosophers MA | 63 | 75.40 | 0 |
| philosophers C | 100 | 92.00 | 0 |
| pipes-notankage NMA | 52 | 0 | 0 |
| pipes-notankage MA | 60 | 13.27 | -0.56 |
| pipes-notankage C | 70 | 16.23 | 0.21 |
| satellite NMA | 89 | 0 | 0 |
| satellite MA | 89 | 28.35 | 1.72 |
| satellite C | 89 | 32.85 | 2.81 |

Figure 4: Performance on Different Domains Using the Different Configurations: NMA—no macro-actions, MA—per-instance Macro-Actions, C—caching Macro-Actions. Mean improvement figures are the mean of the differences between the result achieved with no macro-actions and the chosen configuration: they are calculated only for instances solved in both configurations

| **Domain** | *M.A.vs No M.A.* | | | *Caching vs M.A.* | | | *Caching vs No M.A.* | | |
|---|---|---|---|---|---|---|---|---|---|
| | *n* | *t* | *sig?* | *n* | *t* | *sig?* | *n* | *t* | *sig?* |
| airport | 39 | 5.69 | Yes | 42 | 2.94 | Yes | 39 | 4.22 | Yes |
| depots | 14 | 2.17 | Yes | 15 | 4.70 | Yes | 15 | 3.85 | Yes |
| philosophers | 14 | 1.44 | No | 30 | 5.48 | Yes | 14 | 1.68 | No |
| pipes-notankage | 25 | 2.67 | Yes | 28 | 2.87 | Yes | 24 | 2.24 | Yes |
| satellite | 32 | 2.77 | Yes | 32 | 2.60 | Yes | 32 | 2.74 | Yes |

Figure 5: Performance on Different Domains Using the Different Configurations: M.A. is used as an abbreviation for Macro-Actions; n is the number of problems solved by *both* of the configurations being considered; and t is the t-value obtained from a paired t-test comparing the two. Significance to 95% confidence is shown in the sig? column.

## References

[Botea *et al.*, 2004] A. Botea, M. Muller, and J Schaeffer. Using component abstraction for automatic generation of macro-actions. In *Proceedings of ICAPS-04*, pages 181–190, 2004.

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

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

[Edelkamp and Hoffmann, 2004] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of IPC-4. IPC4 Booklet, ICAPS 2004, 2004.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *IJCAI*, pages 956–961, 1999.

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

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

[McDermott, 2000] D. McDermott. The 1998 AI planning systems competition. In *AI Magazine 2*, pages 35–55, 2000.

[Nau *et al.*, 2003] D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of AI Research*, 20:379–404, 2003.

[Rintanen, 2003] J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proceedings of ICAPS 03*, pages 32–40, 2003.