Extending the Use of Plateau-Escaping Macro-Actions in Planning

Amanda Smith
Department of Computer and Information Sciences,
University of Strathclyde,
26 Richmond Street,
Glasgow, G1 1XH
email: amanda.smith@cis.strath.ac.uk

Abstract

Many fully automated planning systems use a single, domain-independent heuristic to guide search and no other problem-specific guidance. While these systems exhibit excellent performance, they are often out-performed by systems which are either given extra human-encoded search information, or spend time learning additional search control information offline. The benefit of systems which do not require human intervention is that they are much closer to the ideal of autonomy. This document discusses a system which learns additional control knowledge, in the form of macro-actions, during planning, without the additional time required for an online learning step. The results of various techniques for managing the collection of macro-actions generated are also discussed. Finally, an explanation of the extension of the techniques to other planning systems is presented.

Introduction

This document introduces the core ideas in the investigation into the management, and extension of use, of plateau-escaping macro-actions in planning. The ideas are explored in a planner called Marvin (Coles & Smith 2004), which competed in the Fourth International Planning Competition (IPC 4). In the competition Marvin generated *plateau-escaping macro-actions* and used them later in the search process to solve each problem. This investigation extends this approach to allow the caching of these macro-actions for use again in solving future problems, and the issues of library management this entails. Further ideas related to widening the applicability of plateau-escaping macro-actions to other heuristics and other planning technologies are also explored. Many previous approaches to macro-action generation and management require an offline learning step (Botea, Muller, & Schaeffer 2004) (or human intervention (Nau *et al.* 2003)) to generate a set of macro-actions which are then used, unchanged, in solving the remainder of the problem. This approach differs from existing approaches in that it does not require additional offline learning time to generate and filter macro-actions: they are filtered and selected dynamically.

Plateau Escaping Macro-Actions

The underlying search strategy used by Marvin is based on enforced hill-climbing as used in FF (Hoffmann & Nebel 2001). 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: 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 is expensive. 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.

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. Plateau-escaping action sequences are used to construct *plateau-escaping macro-actions*; these can be applied by the planner upon reaching later plateaux. The extraction of plateau-escaping macro-actions from plateau-escaping sequences is computationally inexpensive: the planner simply notes the sequence of actions applied since the start of the plateau. Generation of plateau-escaping sequences is expensive but since the search to find these sequences is being done anyway, to solve the planning problem, no additional search must be done to generate the macro-actions. The impact on planner performance caused by these macro-actions has been evaluated (Coles & Smith 2005).

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

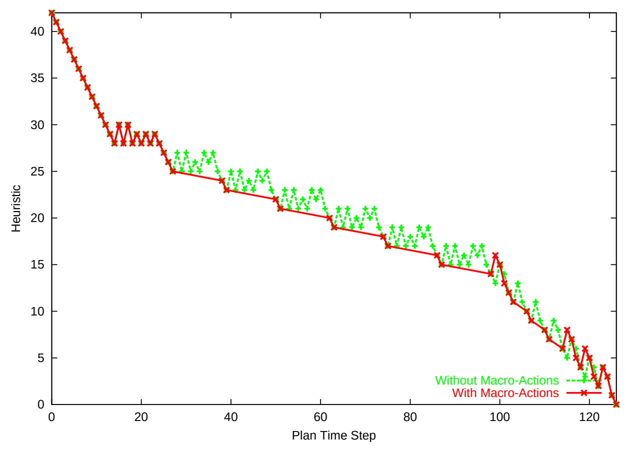

Figure 1: Heuristic Landscape over Makespan, With and Without Macro-Actions

being made into a macro-action. Any independent threads of execution that exist in the plan are separated to produce macro-actions involving as few entities as possible.

Caching Macro-Actions

Having solved one problem in a given domain it would be wasteful to disregard everything that has been learnt in doing so when going on to solve another problem in the same domain. Many of the plateau-escaping macro-actions provide a useful insight into the structure of the search landscape, under the RPG heuristic, across many problems in a given domain not just for a single problem. For example, figure 1 shows the heuristic profile across the solution plan, with and without plateau-escaping macro-actions, to a problem in the philosophers domain taken from IPC 4. The structure is common to all of the problems in the philosophers domain. It can clearly be seen that the macro-actions allow the planner to skip across plateaux exploiting the knowledge gained during previous search to solve a given problem. It would be useful to store macro-actions for use in solving future problems in the same domain. In doing this, however, the issue of increasing the branching factor in the search space by adding macro-actions becomes even more critical.

A large library of macro-actions is generated; a good pruning strategy is therefore essential. If the planner were to store all macro actions generated during the process of solving all problems, and then consider their application at every point during the search, the performance of the planner would almost certainly degrade significantly. One macro-action caching strategy was shown to improve planning performance over a configuration using no macro-actions and over the configuration generating macro-actions on a per-problem basis (Coles & Smith 2005).

Managing a Library of Cached Macro-Actions

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

Each problem is solved only once, this means that the conventional supervised learning techniques used by many systems cannot be used. Due to the nature of the macro-actions used, it is possible to do learning without the need to test the performance of the system with and without each macro-action. The only information available to the system is the number of times the macro-action has been used and the number of problems that have been solved since the macro-action was last used. The strategy for pruning will therefore be based on the usage statistics. If a macro-action is going to be rated based solely on usage statistics there must be a strong reason to believe that use of such a macro-action will improve planner performance. In the general case it may be that a macro-action is used frequently but does not greatly improve search performance: the heuristic may have led search quickly in the appropriate direction without the need for the macro-action. Plateau-escaping macro-actions, however, avoid exhaustive search; it is therefore far more reasonable to use the premise that the application of a plateau-escaping macro-action implies a time saving as a basis for this caching strategy. A similar observation was made by Minton (Minton 1985). The evaluation of the caching strategy itself will determine whether or not this is a reasonable premise on which to base a caching strategy.

The information available from the usage statistics of macro-actions is stored with each macro-action in the library: that is, the number of times the given macro-action has been used and the number of problems solved since the macro-action was last used. Three different approaches to caching macro-actions have been considered, each with various different parameters.

Search-Time Pruning

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

Time-Out Pruning

The motivation for time-out pruning is to reduce the size of the library of macro-actions by removing those macro-actions which have not been useful in solving recent problems. The basis of this strategy is to remove macro-actions that have not been used in solving the last n problems from the library. Some macro-actions are never re-used after being discovered once: such actions will be removed from the library reasonably quickly, thus only being present increasing the branching factor in a few problems.

The success of this strategy will clearly depend on the decided value of n, the *caching interval*. If the caching interval is very short, i.e. n is small, then potentially useful macro-actions may be discarded too hastily; if the caching interval is too large then it is likely that the planner will have to deal with large numbers of non-reusable macro-actions. The value of n that gives the best results varies on a domain-dependent basis; however, in order to create a fully automated system a single value of n must be decided upon. Investigations so far show that value giving the best performance considering all of the domains evaluated is $n = 2$.

Survival of the Fittest

The pruning strategy employed in this approach is to keep only the n most used macro-actions in the library. The larger the value of n is, the more the branching factor is potentially increased; the smaller the value of n, the greater risk of discarding useful macro-actions. The advantage of this approach over the time-out approach is that macro-actions that are useful in many problems will not be deleted from the library simply because they are not used in a (potentially short) run of adjacent problems. This approach also imposes an upper limit on the size of the library meaning that the increase in branching factor is more tightly controlled but a number of the best macro-actions can still be kept.

Investigating the Properties of Useful Macro-Actions

Many systems using macro-actions impose arbitrary limits on both the number of macro-actions to be used by the system at any given time and the maximum length of macro-actions. A flexible system that can dynamically select which macro-actions to use can, however, allow a thorough investigation of the characteristics of useful macro-actions without disregarding certain classes of macro-actions.

Figure 2 shows the mean number of uses of macro-actions of various lengths across a range of domains, most of which are taken from IPC 3 and IPC 4. Many previous approaches have been based on the idea that only macro-actions with short lengths should be used, without necessarily fully investigating usage statistics. It can be seen from the results that plateau-escaping macro-actions of length 2 are used the most frequently but it is interesting to note that macro-actions of other lengths are quite often used. Furthermore logic suggests that use of a longer plateau-escaping sequence will result in a greater performance improvement as more exhaustive search is potentially avoided. Of note is a very large macro-action, of length 71, that was generated in the blocksworld-4ops domain. Further experiments have shown that the use of this macro-action allows three extra problems to be solved within the 30 minute time limit. It is worth noting though that, despite this, performance is degraded slightly on some problems.

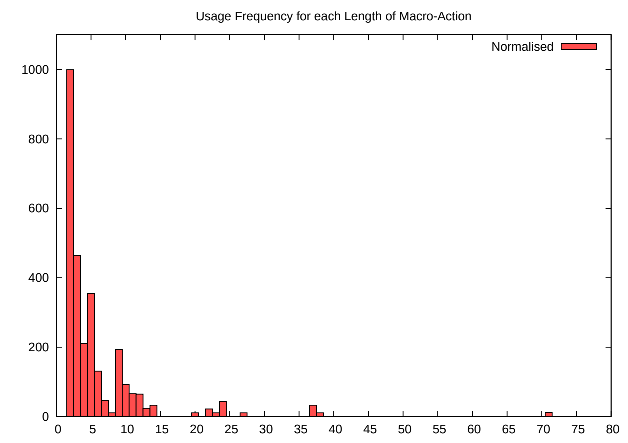

Figure 2: Normalised Usage Statistics for Differing Lengths of Macro-Actions

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

Although this technique has been developed and discussed primarily using enforced hill-climbing under the RPG heuristic the idea can extend to other types of planning technology using different heuristics.

Generating Plateau-Escaping Macro-Actions for the Downward Heuristic

The idea of plateau-escaping macro-actions can be easily mapped to other heuristics used in forward chaining search. Different heuristics give rise to different search landscape profiles; it is, however, the case that all heuristics will have a weakness somewhere[1]. In forward chaining search this can often be characterised by the plateau-like situations observed in FF.

Experiments suggest that the generation of macro-actions can improve the performance of a planner using the downward heuristic to perform enforced hill-climbing. It is however often the case, contrary to the results generated using the RPG heuristic, that a version of the planner generating macro-actions on a per-problem basis generally performs better than a version caching these macro-actions for future use. This is due to the different nature of the two heuristics: the relaxation formed by the RPG heuristic is fixed and will model problems in the same domain in a similar manner. During the calculation of the downward heuristic, however, a step that breaks links based on dependencies is introduced, and the number of dependencies on a given graph-link varies between problems. It is therefore, often the case, that the downward heuristic will give a more different landscape between problems in the same domain, rendering caching of plateau-escaping macro-actions less effective.

[1]That is, of course unless a solution to the problem is to be used as a heuristic which is, of course, not useful: if a solution to the plan construction problem is found it is no longer necessary to search for one.

Macro-actions generated on the same problem, however, appear to give a greater performance improvement than using the equivalent technique under the RPG heuristic.

Further Work

The investigation is to be extended further to explore the wider applicability of plateau-escaping macro-actions. This includes use of plateau-escaping macro-actions, generated under the RPG heuristic, in planners that do not use this heuristic. The motivation for this is that the core hard part of a problem exists when solving it using any approach, and that part does not lie where the relaxed planning graph can accurately model the problem polynomially. Other work to be pursued in conjunction with Adi Botea and Andrew Coles is to investigate the observation that some macro-actions appear to improve search performance due to their role in improving accuracy of the heuristic estimate rather than their application during search.

The extraction of plateau-escaping macro-actions from solution plans for use in various planners is to be considered. Extraction of macro-actions from optimal plans is an extension of this approach: such macro-actions have the potential to reduce the makespan of plans by encouraging the planner to take a route that has previously resulted in an optimal solution.

References

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 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 181–190.

Coles, A., and Smith, A. 2004. Marvin: Macro-actions from reduced versions of the instance. IPC4 Booklet, Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004). Extended Abstract.

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*.

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

Minton, S. 1985. Selectively generalizing plans for problem-solving. In *In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI '85)*, 596–599.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.