Learning to Choose Instance-Specific Macro Operators

Maher Alhossaini
Department of Computer Science
University of Toronto

Abstract

The acquisition and use of macro actions has been shown to be effective in improving the speed of AI planners. Current macro acquisition work focuses on finding macro sets that, when added to the domain, can improve the *average* solving performance. In this paper, we present Instance-specific macro learning. This kind of macro filtering depends on building a predictor that can be used to estimate, for each planning instance, the best subset of a previously collected set of macros to speed up the planning. Learning the predictor is done off-line based on the observed correlation between problem instance features and planner performance in macro-augmented domains. Our empirical results over five standard planning domains demonstrate that our predictors perform as well as the non-instance-specific method that chooses the best-on-average macro, and that there is a chance of improving the performance significantly using instance specific macros. Also in this paper, we tackle the problem of choosing macros from a large set of initial macros as well. We show that in this case approximate methods can produce macro sets that are comparable to the best macro sets.

Introduction

The use of macro action acquisition methods in AI planning has been in the planning literature since its early beginnings (Fikes and Nilsson 1971). They depend on remodeling the planning domains by adding uninstantiated sequences of domain actions, or macros operators, to the domain as an atomic operator. These methods have empirically proved to be useful in improving the planning speed (Coles and Smith 2004). Work that involves automation of macro action generation is common in the planning literature. Many tools have been developed to acquire macros from previously generated plans, such as Wizard (Newton and Levine 2007). There are other tools that generate macros and also use them in their built-in planner, such as Marvin (Coles and Smith 2004) and Macro-FF (Botea et al. 2005). However, most of the existing macro tools help to find macros that work for the domain in general, not focusing on the specific instances within the domain and using previous knowledge. For example, Marvin finds useful macros on-line for the problem instance, but does not save information for future use, and Macro-FF does not remodel the domain for each new problem instances.

Instance-specific macro learning, in contrast, aims to find sets of macros that best improve performance for solving an individual instance based on the analysis of the instance's features. Such instance-specific macro sets are subsets of an original set of macros that were found to be useful for the domain. In rich domains, it may be better to use different macro subsets to solve different instances as opposed to using a fixed set for all instances.

Our approach is to use machine learning to develop a prediction model that relates the problem instance features to the performance of a planner on domains augmented with the different macro subsets. This is done off-line, using a training set of problem instances. Online, the predictor measures the given instance's features and selects a macro subset to be added to the domain to solve the given instance.

In the following, we describe the approaches we took to learn how to predict instance-specific macro actions within a given planning domain. There are two parts of our work:

1. Instead of adding a set of macro operators to the domain permanently, we present a method that only adds macros that are suitable for the individual problem instances. This is done using supervised machine learning techniques and an initial set of macros that are acquired using an existing macro acquisition tool. A prediction model is trained for a given domain such that it maps the domain problem instance's features to different macro subsets based on how well the subset improves the planning speed.
2. One problem that we face in (1) is that as the size of the initial macro set used increases, the learning becomes exponentially harder. To tackle this problem, we use local search in the space of macro sets. This is done using a tool that, for the given problem instances, suggests a macro set, with which we can correlate the instances' features. The prediction model this time is trained to predict what macro subset the local search tool will suggest for the instance rather than what is really best for the instance. The learning is done in a way similar to (1)'s.

Exhaustive Instance-specific Macros

In the planning literature, remodelling the planning domain in general has been the main focus of the macro acquisi-

tion works. The goal of our work, in contrast, is to improve the performance of planners on macro-enhanced domains by trying to predict which macro sets are relevant and useful to a given problem instance based on the features of that instance. To do this, we need to address two issues: a source of macros and the features that will be used for learning and prediction.

To begin our work, we need to have a set of macros to choose from. We tried to obtain our original set of macros from the macro acquisition tool Wizard for some domains. There are two phases in Wizard: a *chunking* phase, which is an intermediate phase that produces unfiltered macros, and a *bunching* phase, that tries to find the best subset of the chunking macros. We used the macros resulting from the chunking phase as our initial set of macros. However, realizing that these initial macro set must be useful in the instance-specific context rather than the domain context, and that Wizard actually aims to find the best average macros set, we have, alternatively, tried to come up with macros manually, such that they can exploit differences between instances.

The second important issue in learning is the selection of problem instance features to be measured and correlated with planner performance. In this work, we have chosen to use straightforward domain-specific features (e.g., in the Logistics domain, the number of cities, number of airplanes, number of packages, etc.). Although this approach is likely to be reflective of underlying problem structure, it has some drawbacks: more insightful problem features must be derived for each domain (which may be a challenge in itself (Carchrae and Beck 2005)).

In the next subsections, we will explain the system in detail, then we will show our experimental results.

System details

At a high-level, this system's design is straightforward and similar to previous work (i.e., (Leyton-Brown et al. 2003)). In an off-line, training phase, we learn a prediction model that relates measures of problem instance features to the performance of the planner in a domain augmented with a given macro set. Online, the features of a new instance are measured and the predictor is used to identify an appropriate macro subset. That subset is added to the domain and the problem instance is solved.

We built two prediction models: the *Direct* model, which predicts the best macro subset directly based on the problem features, and the *Time* model which predicts the run-time of the planner on the problem instance with each of the macro subsets, and chooses the macro subset that it predicts will give the smallest run-time.

We used the off-the-shelf machine-learning tool, WEKA (Witten and Frank 2002), to train the models. We also used the FF planner (Hoffmann and Nebel 2001) in our experiments. Figure 1 shows the training phase for both models. For a given planning domain, the training phase is as follows for the Time predictor:

1. The original domain and planner is provided to Wizard and the output of its chunking phase is obtained. To reduce the subsequent combinatorics, we limit the number of macros to the top n ranked macros, where $n \leq 5$, though none of the problem domains resulted in more than five macros.

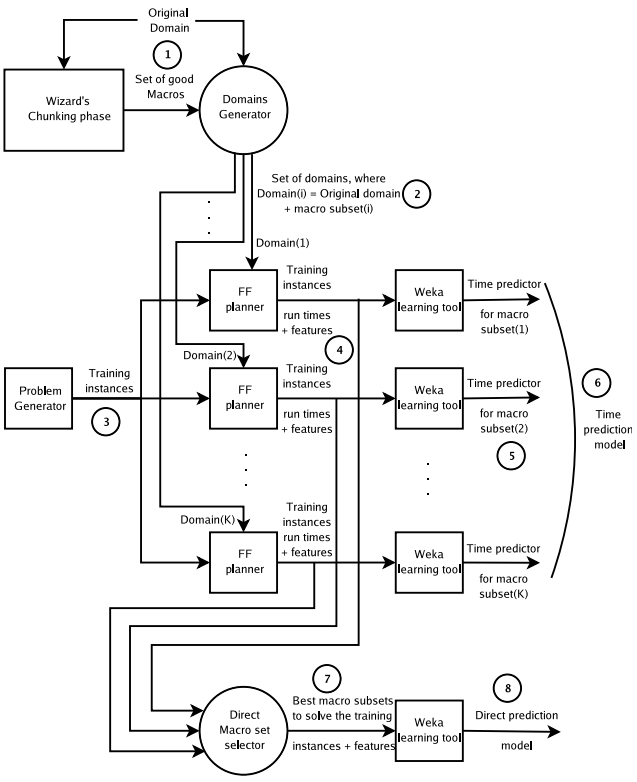

Figure 1: A schematic diagram of the training phase for the Time and Direct predictors.

2. The domain generator creates k macro subsets from the original n macros augmenting the original domain with each subset in turn. This produces k different domains. In this system, we exhaustively generated all $k = 2^n$ subsets.

3. A problem generator for the domain is used to create training instances that span our chosen range of parameter settings.

4. Each of the training instances are solved with the same planner for each of the augmented domains. The run-time for each macro subset and problem domain is recorded.

5. Independently for each augmented domain, a Time prediction model is generated using the M5P (Witten and Frank 2002; Wang and Witten 1997) learning model in the WEKA package. A predictor i attempts to learn to predict the solving time in augmented domain i of problem instances based on their features.

6. The final Time predictor is created by combining each of the individual predictors. When provided with a new instance, the predictor runs each of the individual predictors and chooses the macro subset with the smallest predicted run-time.

The Direct predictor training is done in a similar way, sharing steps 1 through 4 with the Time model and following the subsequent steps, 7 and 8 in in Figure 1, are as follows:

7. The Direct Macro set selector processes the results of running the planner on each training instance with each macro subset. It outputs the problem instance feature measurements and the index of the macro subset which had the lowest run-time.
8. The Direct model is built by WEKA's logistic regression algorithm (Bishop 2006) relating instance feature measurements to macro subset index.

The system produces an on-line predictor of the best macro subset for a given problem instance. The inputs to the predictor are the problem instance features and the output is a new domain consisting of the original domain augmented with macro subset that is estimated to perform best with the instance.

Experimental Results

To evaluate our approach to learning instance-specific macros, we conducted an experiment with five well-known domains: logistics, miconic, blocksworld, mprime, and freecell. These domains vary from easy domains (e.g., miconic) to hard domains (e.g., freecell) with respect to the FF planner (Hoffmann 2001). Each of these domains has a problem generator that is used with difference parameters for the training and testing instances of our experiments.

We used Wizard only in the logistics and miconic domains to come up with the initial macro set. The experiment on these domains were repeated 10 times with different training and test instances. This design allows Wizard to generate different initial macro set in each repetition. In the experiment of the remaining domains, we did not use the Wizard tool to create the macros. We rather generated them manually, trying to find the macros that work well over different instances.

In order to evaluate the performance of our models, we had to measure the performance of other selected models and common macro subsets. We compared the following five models/macro subsets:

1. *Empty* subset: This subset is the original domain, not augmented by any macros.
2. *Direct* predictor (see above)
3. *Time* predictor (see above)
4. *Best-on-average* subset: This macro subset is the one which has the smallest mean run-time on the *training* instances. This macro subset represents the standard approach to macro learning in the literature, and might be thought of as the goal of most of that current macro acquision systems.
5. *Perfect* predictor: This imaginary predictor correctly identifies the macro subset that will have the minimum run-time on the instance. This predictor is created *a posteriori* after having run all the macro subsets on all test instances. The performance of this model is the best that can be achieved by any instance-specific predictor.

Table 1: Average run-times for the models/ macro sets used in the experiment

Domain	Empty set	Time model	Direct model	Best-on-average	Perfect
logistics	6.08	2.94	2.6	2.51	2.32
miconic	1.29	1.48	1.18	1.19	1.18
blocksworld	595.25	54.51	232.69	51.96	30.71
mprime	268.6	209.94	247.6	224.75	51.94
freecell	365.6	347.69	383.84	365.6	104.64

We looked at the mean time taken to solve the test instances using the models/subsets in each of the domains. For the timed-out instances, we registered the cut-off time as their run-time and included that data point in the calculation of the mean. However, the nature of some domains made it very hard for us to come up with a good parameterization that can produce hard-enough instances for the testing. In these domains, namely the blocksworld, mprime, and freecell, we decided (before using the runtimes) to remove the test instance that are too easy, and too hard (instances whose maximum run time is less than a small constant, and instances whose minimum run time is grater than a large constant, where the constants are decided before doing the experiment.)

Table 1 shows the empirical results of our experiments. Our results over five standard planning domains demonstrate that our predictors performs relatively better than unaugmented domain in general. However, results also show that the models perform as well as the non-instance-specific method that chooses the best-on-average macro subset.

We can show using ANOVA tests that in the harder domains (mprime and freecell), the perfect predictor model was significantly better than any other non-instance-specific macro set, that also include the best-on-average macro set. This means that the maximum achievable performance using instance-specific macros (presented by the perfect model) is significantly better than the maximum achievable performance using only fixed macro subset (presented by the best-on-average macro set) in the hard domains. These two domains happened to be the hard domains for the FF planner. This leads us to asking questions about the relation between the domain topology and the instance-specific macros learnability.

Using Local search for Large Macro Sets

The obvious problem with the previous approach is that it becomes impractical when the number of macros in the original macro set is large. This is because the number of runs needed to train the models grows exponentially with the number of macros. However, we need sometimes to consider a large number of macros for the domain. Also, we know that adding too many macros to the domain will negatively affect the performance, and hence we should not include macros that negatively affect performance.

To maintain the instance-specific context that we have fol-

lowed, we need to come up with a method that can find the best macro subset of a huge macros set such that this subset can solve the problem instance as quickly as possible. Obviously, since we do not know any method that can find us the best macro subset unless we try all the combinations, we need to find an approximation. One of the ways that can help us come up with a good macro subset for the instance is to use local search techniques in the space of macro subsets. To do this kind of search, we used an algorithm tuning tool called ParamILS (Hutter, Hoos, and Stutzle 2007).

System details

ParamILS takes as an input: the algorithm to be tuned, the algorithm's parameters, the algorithm's parameters values, and some training and testing instances to be run using the algorithm. It gives as an output: a parameter configuration for the algorithm that improves its performance as much as possible.[1] To use ParamILS in our work, we can think of a macro subset as a vector of binary parameters, where each parameter is 1 if and only if we choose its corresponding macro to solve the problem instances. Our goal then becomes finding the best binary vector value for different feature settings, and training our models with this.

In the following we explain how we use ParamILS to find the macro subsets for the problem instances. The training phase of this learning technique is summarized in the following steps:

1. We are given a big number (N) of macros, and we have M different features settings of the domain instances. A feature setting is an instantiation to each feature variable. It is impractical now to enumerate all of the 2^N macro subsets and run the training instances with them.
2. To train the model, we generate a set of training instances from the domain. Normally, a domain instance generator or benchmark examples can be used for the training.
3. For a possible features setting i, we pick a non-empty set of instances S from the set of training instances whose feature setting is i. If there is no such set, we do not need to include this feature setting in the learning, since we can only learn from the available data.
4. We use ParamILS to find the best parameter setting x_S for the instances in S. The parameter setting represents the macro subset to be chosen for the instances in S and it is represented as a binary vector of size N. So, for example, if $N = 5$, then the parameter setting 01011 means that we chose only the macros number 2, 4, and 5, to add to the domain and run on the planner.
5. After ParamILS finds the appropriate set of macros for the feature setting, we register the features setting i with the chosen macro numbers from the parameter setting x_S in the training file.
6. We repeat steps (3) to (5) for all possible feature settings i. Notice that it is possible to have more than one set S that have the same feature setting. In this case, there will be more that one entry with the same feature setting in the training file.
7. Now we have a training file that relates the problem instance's feature to the macro subset that ParamILS thinks is best for it. So, we feed the training file to a learning tool. The learning tool will use a supervised learning technique to come up with a prediction model using the training data. The prediction model will be used to predict which macro subset ParamILS will suggest based on the given instance's features.

This kind of learning is similar to the direct model approach discussed in the previous section to predict the best macro given the features. We cannot use the time model now because we do not have the time of every training instance on every macro subset.

As a primary step to test the feasibility of such technique, we conducted a small experiment to see how useful the macro subsets that ParamILS suggests are for the whole domain. The experiment was conducted on the mprime domain with a set of hand-written macros. Due to the limited space, we cannot explain all of the results, but the initial results of this experiment show that the difference in mean-time between the real best-on-average macro set and ParamILS suggested macro set was small, which can positively impact the performance when using instance-specific learning.

Conclusion

In this paper, we presented a novel approach to maximize the performance of planners using macro actions. The approach depends on machine learning methods to suggest macro sets based on the measurement of the features of a given problem instance. Off-line, a set of initial macros is provided. All subsets of this initial set are then evaluated by adding them to the original domain and solving a set of training instances. The resulting data is used to learn a predictor that can relate problem instance features to augmented-domain performance. We demonstrate that our models perform as well as the *a posteriori* perfect predictor in some domains. We also show that in 2 of the 5 domains, the maximum achievable performance using instance-specific macros (presented by the perfect model) is significantly better than the maximum achievable performance using only fixed macro subset (presented by the best-on-average macro set.) We also show another approach to solve the same problem with large initial macro set. The approach depends on approximating the best macro set for a problem instance by applying local search in the space of macro subsets. We also mention that the difference in performance between the real best macros set and the the approximate one was found insignificant.

References

Bishop, C. 2006. *Pattern recognition and machine learning*. Springer.

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

[1] ParamILS can also be used to improve other aspects of an algorithm like the solution quality.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Coles, A., and Smith, A. 2004. Marvin: macro-actions from reduced versions of the instance. *International Planning Competition*.

Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3/4):189–208.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2001. Local search topology in planning benchmarks: An empirical analysis. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)* 453–458.

Hutter, F.; Hoos, H.; and Stutzle, T. 2007. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence-Volume 2*, 1152–1157. AAAI Press.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003. Boosting as a metaphor for algorithm design. In *Constraint Programming*, 899–903.

Newton, M. A. H., and Levine, J. 2007. Wizard: Suggesting macro-actions comprehensively. In *Proceedings of the Doctoral Consortium held at ICAPS 07*.

Wang, Y., and Witten, I. 1997. Induction of model trees for predicting continuous classes. In *Proceedings of the poster papers of the European Conference on Machine Learning*, 128–137.

Witten, I., and Frank, E. 2002. Data mining: practical machine learning tools and techniques with Java implementations. *ACM SIGMOD Record* 31(1):76–77.