Learning Instance-Specific Macros

Maher Alhossaini
Department of Computer Science
University of Toronto
maher@cs.toronto.edu

J. Christopher Beck
Department of Mechanical and
Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

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, result in improved *average* solving performance. Instance-specific macro learning, in contrast, aims to build a predictor that can be used to estimate, for each planning instance, a subset of the previously identified macros that is best for a specific problem instance. Based on off-line measures of the correlation between problem instance features and planner performance in macro-augmented domains, we build two such predictors using a standard machine learning approach. Online, the features of the current problem instance are measured and a predictor selects the macros that should be added to the domain. Our empirical results over four standard planning domains demonstrate that our predictors perform as well as the perfect predictor on all domains and never perform worse than any of the other macro-based approaches tested, including an existing macro-learning system. However, a simpler, non-instance-specific method that chooses the best-on-average macro subset over the learning problems performs just as well as our predictors.

Introduction

Macro action acquisition methods are common in AI planning. They depend on finding a set of macro actions, that is, sequences of domain actions, and adding them to the domain as atomic actions. This remodeling of the domain has been shown to be effective in improving the speed of search (Coles, Fox, and Smith 2007; Botea et al. 2005; Newton et al. 2007). The work done so far focuses on finding the macro set that can improve average problem solving performance in a given domain. 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, and when we have a large set of potentially useful macros, it may be better to use different subsets to solve different instances as opposed to using the whole set for all instances. For any given instance, we conjecture that a large number of macros and/or irrelevant macros can affect the performance negatively, since they increase the branching factor unnecessarily.

Our approach is to use machine learning to develop a prediction model relating 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. We experiment with three different predictors. Online, when presented with a problem instance, the predictor measures the instance's features and selects a macro subset to be added to the domain to solve the given instance.

The novel aspect of our approach is that it focuses on the planning instance rather than the planning domain. We are learning how to best remodel the domain for a given instance based on its features as opposed to finding a fixed set of macros applied to all instances in the planning domain. We have three reasons to believe that our approach may be successful. First, adding too many macros to a domain, even if they individually might improve search on every instance, reduces the planner performance due to an increased branching factor. If we manage to find a small macro subset that is good enough, we may improve the performance just because the number of operators is reduced. Second, our method builds on the current method of learning as it uses macros suggested by a current system, Wizard (Newton and Levine 2007), as input. Instead of adding all the good macros to any problem from the domain, we add only those that appear relevant to the problem instance. Finally, and more broadly, real planning problems are large and complex. For such problems, humans tend to consider different approaches depending on the instance and the extent to which it is similar to other problems solved in the past. The problem solver should be able to learn to remodel the problem domain based on experience with similar problem instances. We also conjecture that, in the traditional macro learning approaches, it is usually not the case that many macros are used in the plan for a given instance. Therefore, there may be scope for improvement by identifying a few promising macros for an instance.

This paper makes the following contributions:

1. We show that using a particular macro subset to solve all instances of the domain is sometimes less effective, on average, than using a macro subset that is predicted to suit

the problem instance.

2. We introduce a way to learn a predictor of the best subset of macro operators using planning instance features.
3. We demonstrate that our approach can build on and outperform an existing macro acquisition tool.

Literature Review

Macro acquisition has a long history in planning, dating back as far as the the work of (Fikes and Nilsson 1971). The approach primarily depends on the fact that plans for different problem instances in a domain often contain common small sequences of actions. These action sequences can be extracted and, in generalized form, added to the domain to enhance the search when solving new problems. This approach can be beneficial because the search does not have to rediscover subsequences of actions if they can be applied for the new problem instances. The drawback, of course, is that macro actions that are not relevant for a given problem instance can act to slow performance as they expand the branching factor. Finding the right set of macro actions, therefore, is important if search is to be improved.

Wizard (Newton and Levine 2007; Newton 2008), is a recent macro acquisition system, which has shown promising results over a number of planning domains and planners. For a given planner and problem generator, Wizard identifies and improves macro actions with a two-phase approach:

1. Chunking: Wizard generates small problem instances and solves them with the specified planner. Using a lifting operator, promising subsequences are extracted from the plans. A genetic algorithm is then used to modify starting macros with each generation being evaluated based on problem solving performance on a set of ranking problems. The result of the chunking phase is a ranked list of macros of which the top ones are returned based on a user specified quality threshold.
2. Bunching: Starting with the output of the chunking phase, the bunching phase uses a genetic algorithm to search through macro subsets. As in the chunking phase, each subset is evaluated using ranking problems and the final output is the highest-rated macro subset.

Experimental studies show an improvement in the problem solving ability when the new macro subset is added to the domains and evaluated on a set of test problems. Importantly, this result is independent of both the domain and the planner. That is, Wizard can successfully find useful macro subsets for different combinations of planner and domain rather than being a system which is specific to one planner or one domain.

In other work, Botea el al. (Botea et al. 2005) presented Macro-FF, a planner built on top of the FF planner. Macro-FF learns macro operators off-line and uses them during the planning process. The approach automatically analyzes the planning domain to discover abstract components, generates macro operators using the components, and then filters and ranks the macros using training instances. A set of macro operators is initially generated by searching the space of operators, such that the operators meet some constraints that consider the abstract components' structures. Then, the macros are filtered and ranked based on how likely they appear in the plans of solved problems.

In our work, we follow an approach similar to that of Leyton-Brown and co-authors, who have built off-line predictors for the winner determination problem (Leyton-Brown et al. 2003) and SAT solving (Xu et al. 2007). In both cases, the authors emphasize the fact that there is, usually, not a best algorithm for a given problem instance, and that relative algorithm performance varies on different instances. The authors show that a strategy that chooses an algorithm based on the features of the problem instance (and off-line learning) provides better performance than the winner-takes-all approach which chooses the one algorithm that is best, on average, over the set of training instances. The core of the work is using machine learning to build a multiple regression-based predictor that relates algorithm performance to problem instance features.

Roberts et al. in (Roberts et al. 2008), have done similar learning work in planning, though not in the context of macro learning. A system is developed to predict which of 28 different planners will succeed in solving a given problem instance and which of them will have the best time based on measuring problem instance parameters. The system used the Waikato Environment for Knowledge Analysis (WEKA) machine learning tool (Witten and Frank 2002) for the training and prediction. Thirty-two different techniques available in WEKA are tried with every planner to learn which works best with each planner. The problem instance parameters were domain independent, not referring to particular objects or actions of a domain but rather to structural characteristics such as the number of operators and predicates, the arity of predicates, the number of predicates in preconditions, etc. The approach performs better on average than any individual planner and than choosing random planners taken from a set of top performing planners.

Approach

The main goal of this work 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. Rather than augmenting a domain once with a set of macros that work well on average over a training set, we want to more specifically identify macros that should be added to the domain for a given problem instance. To do this, two issues must be addressed: a source of macros and the features that will be used for prediction. We address each of these in this section before presenting our approach in detail.

- A source of macros: In order to evaluate the useful macros for a problem instance, we need to have a set of macros to choose from. Furthermore, these macros need to be of reasonable quality since our approach cannot perform better than the "perfect" predictor that can find the best macro subset for any given instance. Our approach is to start with the macros produced by the chunking phase of Wizard. As noted above, this phase finds a set of individual macros that are highly ranked based on average plan-

ner performance on a domain that is augmented with the macro.

- Problem instance features: A more critical issue is the selection of problem instance features to be measured and correlated with planner performance. Unlike Roberts et al. in (Roberts et al. 2008), we have chosen to use domain-specific features (e.g., in the Logistics domain, the number of cities, number of airplanes, number of packages, etc.). We believe that the values of such features are much more likely to be reflective of underlying problem structure than the domain independent features used by Roberts et al: we believe that it is much more likely that our features reflect something meaningful that can be learned. However, there are obvious drawbacks to this approach: meaningful problem features must be derived for each domain (which may be a challenge in itself (Carchrae and Beck 2005)), a predictor must be learned for each domain and planner, and our domain-independence is somewhat questionable. We return to this issue in the Discussion section below.

At a high-level, our system 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.

System Details

We build 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 for a given problem instance for each macro set. The Time model is composed of a number of small predictors, one for each macro subset, that individually try to predict the time needed to solve the instance using the corresponding macro subset. The macro corresponding with the smallest predicted run-time is chosen.

We also use the WEKA machine learning tool. For our experiments, we used two learning models from the built-in learning models of WEKA: the M5P learning model for our Time predictor, and the logistics regression for our Direct predictor.

Figure 1 shows the training phase for both models. For a given planning domain and planner, the training phase is as follows for the Time predictor (the numbering in Figure 1 corresponds to these steps):

1. The original domain and planner is provided to Wizard and the output of its chunking phase is captured. To reduce the subsequent combinatorics, we limit the number of macros to the top n ranked macros where n is a relatively small number. We use $n = 5$ in our experiments, though none of the problem domains resulted in more than five macros, and so this threshold was not active.
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 our experiments, we exhaustively generated all $k = 2^n$ subsets.

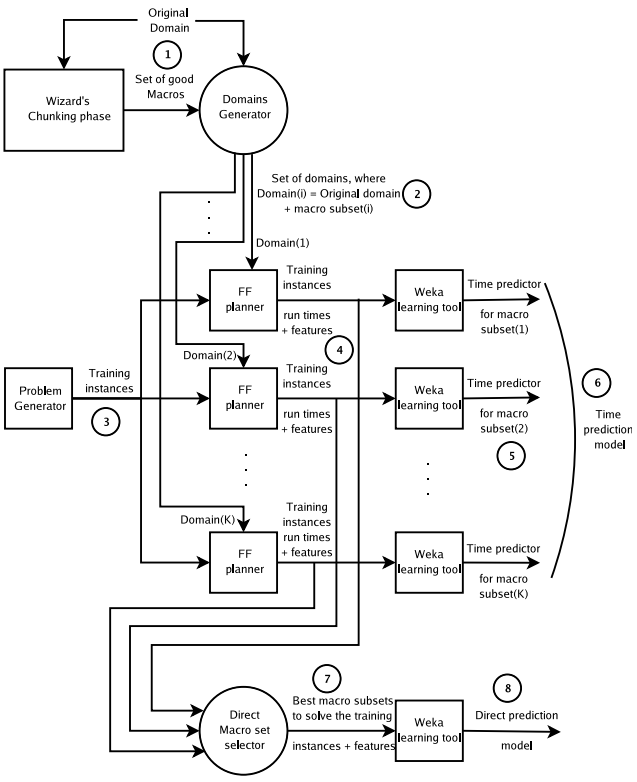

Figure 1: A schematic diagram of the training phase for the Time and Direct predictors.

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. In our experiments, we used the FF planner (Hoffmann and Nebel 2001) as shown in Figure 1.
5. Independently for each augmented domain, a Time prediction model is generated using the M5P 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, breaking ties randomly.

The Direct predictor is built in much the same fashion and, indeed, steps 1 through 4 for the Time model are identical. The subsequent steps, 7 and 8 in in Figure 1, are as follows:

Table 1: Selected features of the four domains used in the experiments.

Domain	Features	# of Instances
Logistics	cities: $\{3, 5, 7\}$ airplanes: $\{1, 3, 5, 7\}$ locations per city: $\{3, 5, 7\}$ packages $\{11, 16, \ldots, 61\}$	Training: 1980 Testing: 396
Ferry	locations $\{5, 10, \ldots, 50\}$ cars $\{10, 20, \ldots, 100\}$	Training: 500 Testing: 100
Gripper	balls $\{1, 6, \ldots, 401\}$	Training: 405 Testing: 81
Miconic	floors $\{2, 7, \ldots, 97\}$ passengers $\{1, 21, \ldots, 401\}$	Training: 2095 Testing: 419

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, with ties broken randomly.
8. The Direct model is built by WEKA's logistic regression algorithm relating instance feature measurements to macro subset index.

Conceptually, the system will produce a ready-to-use 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 the macro subset that is estimated to perform best with the instance.

Experiments

In order to evaluate our approach to learning instance-specific macros, we experiment with four well-known domains: logistics, ferry, gripper, and miconic. In this section, we discuss the domains, the domain features, and present the experimental details.

Experimental Domains

We have chosen four domains for our experiments: logistics, ferry, gripper, and miconic. These domains vary from small domains with a small number of operators (e.g., gripper) to large domains with many operators and a variety of instances (e.g., logistics). Each of these domains has a problem generator that is used both internally for Wizard (i.e., to generate its seeding and ranking instances) and for the training and test instances for our experiments.

Table 1 presents the features and feature-values for each domain. Note that these features are also the input parameters to the corresponding problem generators. Using the same characteristics as features and as generator parameters again raises the issue of feature engineering and the objection that real problems do not come with convenient problem generators attached to them. Again, we defer addressing these issues to the Discussion.

Experiment Details

The experiments were conducted on a Beowulf cluster with each node consisting of two Dual Core AMD 270 CPUs, 4 GB of main memory, and 80 GB of disk space. All nodes run Red Hat Enterprise Linux 4. The programming language that we used to write all of our code was C.

For each domain, we ran the experiment 10 times. In each repetition, we generate an initial set of macros using the chunking phase of Wizard and produce domains augmented with each subsets of the initial macros. As noted above, this never exceeded 32 domains. Then, using a problem generator with varying parameter settings, we generate a set of training and testing instances. For every parameter setting, we generate one test instance and five training examples. The total number of instances for each domain is displayed in Table 1. Note that these are the number of instances for a single repetition. Over all repetitions, the number of training and test instances is ten times larger.

Each training instance is solved with each augmented domain by the FF planner with a 600 CPU second time-out. Any instances that failed to solve within that time-limit are treated, for the purposes of learning, as if they had been solved with a run-time of 600 seconds.

For the Time predictor, we used WEKA's M5P decision tree learning algorithm to learn a predictor for each macro subset as described above. For the Direct predictor, we used WEKA's logistic regression model.

In order to evaluate the quality of the predictions from our models, we also ran FF on every test instance with every macro subset. This methodology, for example, allows us to *a posteriori* create a perfect model that is able to infallibly choose the best macro subset for each test instance.

The experiment is repeated 10 times for each domain. In each repetition, we generate a new macro set from Wizard.

Results

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

1. *Perfect* predictor: This predictor infallibly identifies the macro subset that will have the minimum run-time on the given instance. We create this predictor *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 predictor.
2. *Direct* predictor (see above)
3. *Time* predictor (see above)
4. *Empty* subset: This subset is the original domain, not augmented by any macros.
5. *Full* subset: This subset is all the macros identified by Wizard's chunking phase. It is the original macro set used as input to our off-line learning mechanisms.
6. *Best-on-average* subset: This subset is the one which has the smallest mean run-time on the *training* instances. This is not an instance-specific macro subset but rather represents the standard approach to macro learning in the literature.

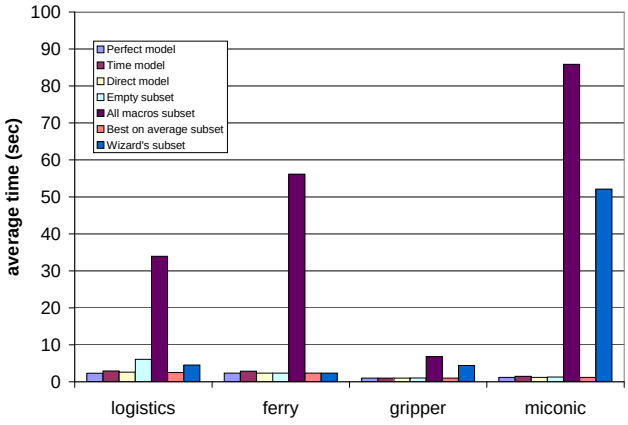

Figure 2: The mean time taken to solve the testing problems in each of the domains.

7. *Wizard*'s macro subset: This subset is the one chosen by Wizard, using its default parameters, after both the chunking and bunching phases.

The experiment on each domain was repeated 10 times with different training and test instances. This design means that, with the exception of the Empty subset, each of the models/subsets above potentially identifies a different subset for each repetition: the Best-on-average subset in repetition 1 may not be the same as the Best-on-average subset in repetition 2.

Figure 2 shows the mean time taken to solve the test instances using the models/subsets in each of the domains for the 10 repetitions. For the timed-out instances, we registered 600 seconds as their run-time and included that data point in the calculation of the mean. The figure shows the clear result that using all the macros of the original set (Full) has a substantial negative effect on the performance. It also shows that the Direct predictor model was equal to or better than the Time model. Our models, the Perfect model, and the Best-on-average subset display very similar mean run-times.

Figures 3, 4, 5, and 6 show the fraction of test problems solved over time for each of the models/subsets in each of the domains. Note that we used difference ranges in the x-axis in these graphs to clarify the differences between the sets. As in Figure 2, these are aggregate results over the 10 repetitions of each domain. For a given domain and model/subset, we calculate the mean fraction of solved instances over each repetition.

Figure 3 shows, unlike the other three figures, that there is a clear order in performance: the Perfect model, the Best-on-average and Direct model, the Time model, Wizard, the Empty subset, and finally Full subset. In Figure 4, the difference between the data sequences of the top models/subsets was not as clear, except for Time model which was slightly worse than the rest. In Figure 5, Wizard's subset was worse than the other models/subsets (except Full), the empty subset was slightly worse than the top four models/subsets, and the difference between the top models/subsets was, also, not clear. In Figure 6, Wizard was worse than the top five, the Time model was slightly worse than the top four, the Empty set was slightly worse than the top three, and the difference between the remaining models/subsets was marginal. One common result in these graphs is that the full subset was the worst.

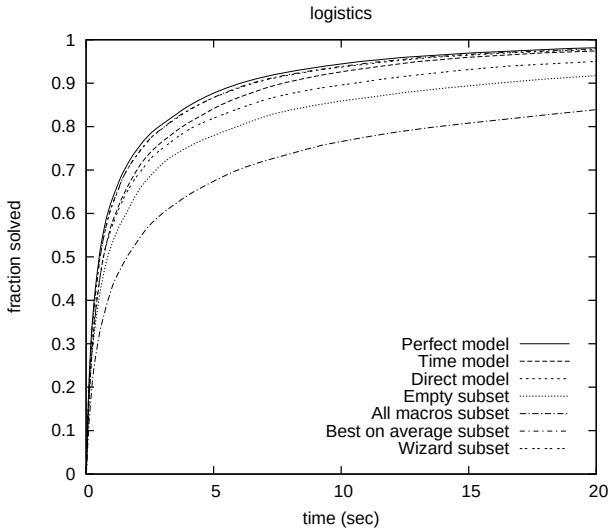

Figure 3: Mean solubility vs. time for each of the seven models/subsets in the logistics domain.

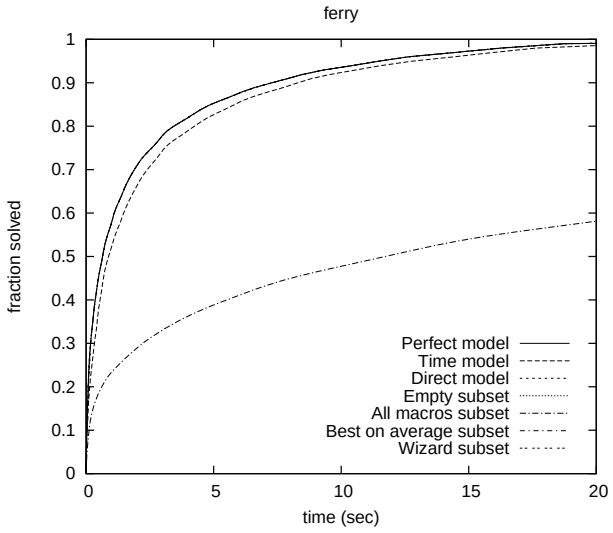

Figure 4: Mean solubility vs. time for each of the seven models/subsets in the ferry domain.

To test the differences between the mean run-times of each of the chosen models and macro sets, we performed an ANOVA and Tukey's Honest Significant Difference (HSD) test (Tukey 1973) using the statistical package, R (R De-

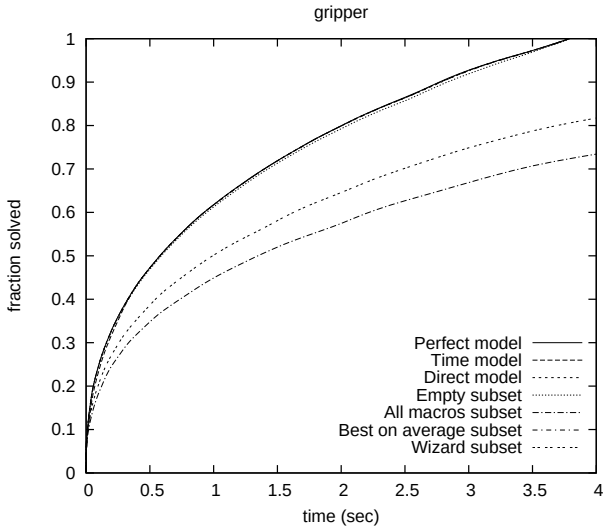

Figure 5: Mean solubility vs. time for each of the seven models/subsets in the gripper domain.

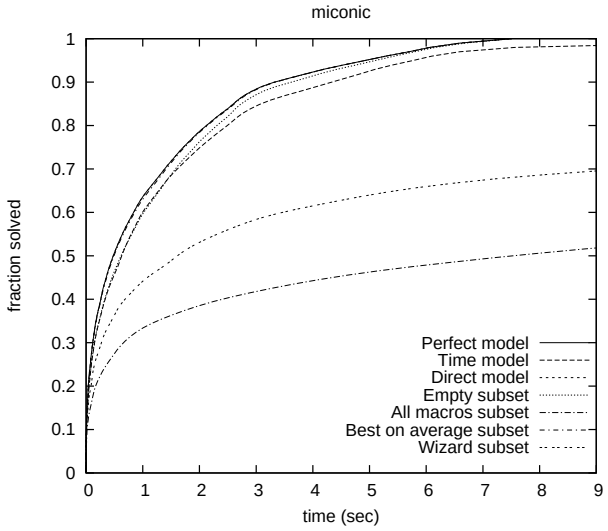

Figure 6: Mean solubility vs. time for each of the seven models/subsets in the miconic domain.

velopment Core Team 2006). The ANOVA indicated that the model parameter was a significant factor. Table 2 shows the significant differences at $p \leq 0.005$ between all pairs of subset/models. The Tukey HSD test corrects for the impact of multiple pair-wise statistical comparisons. The data series of the models/subsets were the run-times taken to solve every test instance with each model/subset on every repetition. The table shows the number of domains in which the models/macro subsets were significantly better than the other models. The entries of the form (+x,-y) mean that the model/set that corresponds to the row is significantly better than the model/set that corresponds to the column in x domains while being significantly worse in y domains. The difference between the sum of x and y and 4 (i.e., the total number of domains) indicates the number of domains with no significant difference between the models/sets.

It can be seen that the Perfect model is significantly better than the Empty, Full, and Wizard subsets in one, four, and two domains, respectively. Naturally, no model was significantly better than the Perfect model. The Direct model and the Best-on-average subset were not statistically significantly different than the Perfect model on any domain. The Time model performed similarly to the Direct model except that it was not significantly better than the Empty subset in any domain, whereas Direct significantly out-performed the Empty subset on one domain. The Empty subset was significantly better than the Full subset in three domains and significantly better than Wizard in two domains. Wizard's subset was better than the full subset in all four domains.

Discussion

Our experimental results demonstrate that the Direct and Time predictors perform as well as the Perfect predictor and better than Wizard. A more even-handed examination of the data, however, suggests that we have not found convincing evidence for our belief that domain remodeling via instance-specific macros can improve planner performance. Table 2 also shows that Best-on-average, the macro set with the lowest mean run-time on the training instances, also performs just as well as the Perfect model and also better than Wizard. Furthermore, the Perfect model is only significantly better than the Empty model on one of the four domains. It appears, therefore, that our experiments suffer from a ceiling effect: there is little room to significantly improve planner performance by choosing a subset of the original set of macros. This could be due to the starting macro set being of too low a quality, to the domains not being amenable to macro-augmentation, or, most likely, to test problems being too easy. A primary direction for future work, therefore, is to understand the reasons underlying the results we have observed.

The comparison between Best-on-average and Wizard is also a bit unfair. Recall that the bunching phase of Wizard begins with the same macros as the Best-on-average and applies a genetic algorithm to search within the space of macro subsets. Best-on-average performs an exhaustive test of *all* subsets. It is to be expected that the increased computation of Best-on-average would lead to better performance. While we believe that Best-on-average is a reasonable approach, since macro learning is an off-line process and at least from the domains tested there do not seem to be a large number of high-quality candidate macros, it is also expected that Wizard will scale much better as this candidate list grows.

The issue of problem instance features has been alluded to above. Knowledge-based problem-solving techniques such as explored here but, of course, dating back to at least the 1970s (Simon 1973), require some problem structure that can be recognized and used. We do not believe that domain-level characteristics like the number of operators or their arity reflect a rich enough structure to be usefully exploited.

Table 2: Statistical differences between pairs of selected macro subsets and models of the experiment.

	Perfect	Time	Direct	Empty	Full	Best-on-average	Wizard
Perfect		(+0,-0)	(+0,-0)	(+1,-0)	(+4,-0)	(+0,-0)	(+2,-0)
Time			(+0,-0)	(+0,-0)	(+4,-0)	(+0,-0)	(+2,-0)
Direct				(+1,-0)	(+4,-0)	(+0,-0)	(+2,-0)
Empty					(+3,-0)	(+0,-1)	(+2,-0)
Full						(+0,-4)	(+0,-4)
Best-on-average							(+2,-0)
Wizard							

While such measures may embody some indication of the possible aggregate complexity of a domain (e.g., the number of different types of objects that may affect each other), the characteristics of a given instance will vary widely depending on the number of objects of each type that are present. In short, we are looking for exploitable structure where we think it might exist.

However, this approach, as well as some of the decisions made in this paper, introduce a number of weaknesses to be addressed.

- Is our approach domain independent? Our overall approach can be applied to any domain, provided that there is a source of problem instances and a defined set of features. The requirement for a feature set does increase domain dependence but we do not believe that this is on the same scale as the knowledge and control rules that have traditionally informed domain dependent planners.
- Where do the features come from? Once features are required, it is necessary to engineer them. We believe this is a weakness of "high-knowledge" problem solving, in general (Carchrae and Beck 2005) and, indeed, some efforts in this direction in other problem domains are extensive (Xu et al. 2007; Leyton-Brown et al. 2003). However, in planning, at least at this stage in its development, it does not seem a significant overhead beyond the domain definition itself: a domain modeler is likely to be able to easily identify interesting instance features. Furthermore, automated feature generation, at least for basic features, may be feasible by examining the differences among problem instances (e.g., types with a varying number of objects in different problem instances).
- Is basing the features on problem generator parameters problematic? Our choice to using problem generator parameters as features is natural and supports our claim above that domain modelers are likely to be able to generate features. These parameters are exactly what the author of the generator believe would create instances with a variety of characteristics and, therefore, they represent just the type of knowledge we would like to exploit. While real-world problems do not come with their own generators, it is unlikely that we are going to be able to scientifically study the solving of multiple instances in a domain without the ability to create such a generator. Finally, some readers may argue that our approach is hiding structure in a problem instance based on the generator parameters only to tell the learner exactly where to look to uncover the "hidden" structure. As implied above, we believe that the process of identifying features is fundamentally related to identifying useful problem generator parameters. To understand a new domain and to model it or develop planners that can solve it, a scientifically minded modeler will necessarily make and test hypotheses about what aspects of the domain make the problem challenging. It is precisely those hypotheses that are supported that result in both interesting generator parameters and promising problem features. Rather than hiding and discovering artificial structure, both problem generator parameters and instance features represent our best understanding of the factors responsible for problem difficulty.

Conclusion

In this paper, we presented a novel approach to macro acquisition that 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 candidate macros is produced by the first phase of an existing macro acquisition system, Wizard. 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 produce a predictor to related problem instance feature measurements to augmented-domain performance. Using the FF planner and the machine learning tools available in the WEKA system, we demonstrate that our models perform as well as the *a posteriori* perfect predictor. However, we also show that the standard approach of identifying the macro subset that performs best on average over the training set also performs as well as the perfect predictor. We conjecture that more challenging test instances may better reveal performance differences between the models.

Acknowledgements

The authors wish to thank M.A.H. Newton for making the Wizard source code available and for helpful discussions.

References

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.

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. I.; Fox, M.; and Smith, A. J. 2007. Online identification of useful macro-actions for planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 07)*.

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.

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*.

Newton, M.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the ICAPS*.

Newton, M. 2008. *Wizard: Learning Macro-Actions Comprehensively for Planning*. Ph.D. Dissertation, Department of Computer and Information Science, University of Strathclyde, United Kingdom.

R Development Core Team. 2006. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Roberts, M.; Howe, A. E.; Wilson, B.; and desJardins, M. 2008. What Makes Planners Predictable? In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*.

Simon, H. A. 1973. The structure of ill-structured problems. *Artificial Intelligence* 4:181–200.

Tukey, J. 1973. *The problem of multiple comparisons.* Princeton University Princeton, NJ.

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

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2007. SATzilla-07: the design and analysis of an algorithm portfolio for SAT. In *Principles and Practice of Constraint Programming (CP-07)*, 712–727.