Instance-Specific Remodelling of Planning Domains by Adding Macros and Removing Operators

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

We propose an approach to remodelling classical planning domains via the addition of macro operators and removal of original operators either for the domain as a whole or instance-by-instance. For the latter remodelling, we train a predictor to choose the best reformulation of the domain based on instance characteristic. In the domain level remodelling, we try find a fixed remodelling that works best on average over our training set. Operator removal does not generally preserve solubility and proving solubility preservation of domain models is PSPACE-complete. So we use an approach that uses training instances to empirically estimate the probability of solubility preservation and maintains a minimum value of that probability on the training instances. We show that the instance-specific approach outperforms the traditional best-on-average macro-only remodelling approach in 9 out of 14 cases of domain/macro-source combinations, and that it can outperform fixed domain-based models generated with existing macro learning tools.

Introduction

A central idea in knowledge-based problem solving (Simon 1973) is the interleaving of searching for a solution within a particular problem model and changing that model via remodelling. Simon proposes a remodelling component based on a *retrieval system* that modifies the problem model using previous knowledge and generates new modelling alternatives based on features of the current problem model and state of the search. Much of automated planning has focused only on search in a fixed model. Here, we focus on the remodelling aspect. While we do not remodel the problem *during* search, as proposed by Simon, we use a learned predictor to add macro operators and remove original domain operators before search on a given instance or domain.

Our experimental results using a state-of-the-art STRIPS planner across 14 domain/macro-source combinations indicate that our instance- and domain-level remodelling techniques both significantly out-perform an existing macro-only learning approach as well as an exhaustive best-on-average macro-only approach.

Macros are sequences of operators that act as a single operator in a planning domain. Traditionally, they are added to the domain as basic operators to improve the performance since they act as shortcuts to deeper states in the search. Based on an input set of macros and a set of learning instances, we use machine learning techniques, offline, to learn a predictor that specifies the subset of macros and original domain operators that should make up the domain operators for a given problem domain or instance. For domain-level remodelling, the proposed set is the "best on average" over the learning instances. Online, we simply replace the original domain with the proposed set. For instance-specific modelling, offline we find the best remodelled domain for each instance and use the features of each problem instance and its corresponding best domain as input to a machine learning technique. We, thereby, learn a predictor that maps problem instance features to remodelled domains. Online, we measure the features of a new instance and use the predictor to identify the domain to use for the instance.

As we are removing operators, the solubility preservation of the remodelled domain is not guaranteed: there may be no valid plan in the remodelled domain while there was one in the original domain. However, deciding whether an operator is necessary to solve a problem instance, in general, is PSPACE-complete (Ferrara, Liberatore, and Schaerf 2005). To tackle this issue, we limit operator removals during training to maintain a minimum level of expected solubility preservation for each accepted domain model: if the domain model fails to solve a given proportion of the solvable training instances, we discard it. We discuss solubility preservation in more depth below.

Background

Representing a problem is important for solving it. One well-known representation language for classical planning is STRIPS (Fikes and Nilsson 1971). Conventionally, a STRIPS planning problem is represented by a planning domain and a planning instance from that domain. The planning domain contains a description the environment: the predicates that describe a state and the operators that are used to transition between states. The planning instance is a description of a particular problem: it describes the facts in the initial state and the goal. The purpose of planning is to find a plan, composed of instantiated operators, that leads us from the initial state to the goal.

More formally, a planning problem is a tuple $\mathcal{P} = (D, i)$,

where D is the domain and i is the problem instance (Ghallab, Nau, and Traverso 2004). Often, the same domain is used with different problem instances. The *domain* D is a set of predicates Ψ, and a set of operators O: $D = (\Psi, O)$. A predicate is a relation with a sequence of arguments. A predicate $p \in \Psi$ can be *grounded* (or *instantiated*) into an *atom*, α, using a *substitution*, θ, that replaces all of its variable arguments by constants: $\theta(p) = \alpha$. The substitution can also be applied to a set of predicates to produce a set of atoms. An operator $o \in O$ is a tuple: $o = (pre_o, add_o, del_o)$, where pre_o is the precondition predicates set, add_o is the add predicates, and del_o is the delete predicates. An *action*, $a = (pre_a, add_a, del_a)$, is an instantiation to an operator o with a substitution θ_a. So, $pre_a = \theta_a(pre_o)$, $add_a = \theta_a(add_o)$ and $del_a = \theta_a(del_o)$. A *state* is a set of atoms that are considered true in the state. The atoms that are not in a state are assumed to be false. An action a is applicable in a state s if $pre_a \in s$, and the resulting state from applying a to s is obtained by the function: $a(s) = s \setminus del_a \bigcup add_a$. The problem instance i is a pair: $i = (s_0, g)$, where s_0 is the initial state, and g is the goal partial state. A goal state for i is one that is a superset of g. A *plan* is a sequence of actions. To solve i, a plan π must transform s_0 to a goal state. Such plan is called a *solution plan* to i.

One common approach to domain remodelling is to add macro operators. A *macro operator* is a sequence of operators that act as single operator. A *sound* macro is one whose application in a state is always equivalent to the ordered application of its constituent operators. Tools such as Wizard (Newton 2008) and Macro-FF (Botea et al. 2005) acquire macro operators and add them to the domain as new normal operator, solving all subsequent problem instances with the augmented domain. This is a domain-level remodelling. Macros have also been used in different contexts. For example, the Marvin planner (Coles and Smith 2004), a modified version of the FF planner, learns macros during the local search phase and uses them in later stages to escape plateaux.

Another remodelling approach is to remove irrelevant operators or actions as they can increase the number of states explored. Even planners with the ability to detect irrelevant actions can sometimes fail to find them, and therefore a specialized tool that detects and removes these operators can be useful (Nebel, Dimopoulos, and Koehler 1997). Haslum and Jonsson, for example, identified and removed actions that can be replaced by other actions in the plan (Haslum and Jonsson 2000). In a preprocessing step, they prove that an action is subsumed by a sequence of actions that do not contain it, and then they remove the action. This method is solubility-preserving, but it sometimes requires a lot of time for the preprocessing.

While macros work as shortcuts in the search space, they increase the branching factor and potentially make some operators redundant. Removing the redundant or irrelevant operators, in contrast, decreases the branching factor but it is difficult to find operators that are safe for removal for all instances in a domain. Therefore, we believe the combination of macro addition and operator removal in an instance-specific context is a valuable direction for investigation.

One of the few papers that combines macro addition with the operator/action removal is (Chrpa 2010). Macros are extracted based on action dependencies and added to the domain. Then some *actions* are removed by modifying their corresponding operators so that they can only be instantiated under restricted conditions. Operators whose ground actions have preconditions (effects) that appear only in the initial state (goal) of training instances plans are replaced by copies that are only applicable in the initial state (goal). This method can reduce the branching factor, speeding up planning, but it is not clear whether such operators appear frequently in the planning domains.

We were inspired partially by the large algorithm configuration literature such as ParamILS (Hutter et al. 2009) and, in fact, in a previous work, we used a similar approach to perform an instance-specific macro-based remodelling (Alhossaini and Beck 2012). In (Alhossaini and Beck 2012) we present a method for instance-specific domain remodelling where the domain is remodelled after observing, online, the instance to be solved. The approach uses a combination of offline parameter optimization and machine learning. It was found that the potential of improving planning speed using such context significantly exceeds that using domain-level remodelling. The instance-specific approach appears particularly appropriate for operator removal as the characteristics of a given problem instance may result in unnecessary operators (e.g., the fuel operator, 'donate', in mprime domain instances where there is plenty of fuel). When we add macros, more operators may become removable in the instance-specific context.

The use of machine learning to choose a solving technique is not new in planning. For example, it was used to train a predictor to chooses the best heuristic during the search (Domshlak, Karpas, and Markovitch 2010), and was used to generate rules to choose the best configuration of a planner (Vrakas et al. 2003).

Definitions

Given a planning domain and a set of sound macro operators, one can make a new *domain model* by adding macros and removing operators. However, remodelling the domain this way can make solvable problem instances unsolvable if operators necessary to reach the goal are removed.

There appears to be no agreement in the literature on a term that describes the ability of a domain model (or generally, a reduction) to preserve the solubility of instances in planning domains. For example, Chen and Yao used the term 'completeness-preserving' to describe their state-space reduction in planning (Chen and Yao 2009). Nebel, Dimopoulos, and Koehler used the term 'solution-preserving' to refer to the same property (Nebel, Dimopoulos, and Koehler 1997). Others like Chrpa's did not use a specific term for the property (Chrpa 2010). We choose the term *solubility-preserving*, as we think it is easier to understand and more accurate. We use the term to describe the property of a domain model on single instance. To describe the property for a reformulation to be solubility-preserving on a randomly chosen instance, we use another term: *coverage*.

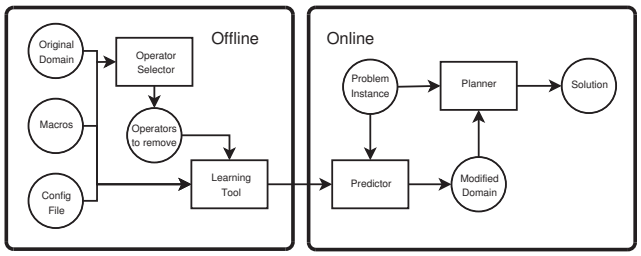

Figure 1: The system framework, with macro addition and operator removal.

A problem instance, i, is *solvable* in the domain model d if there exists is a plan constructed from the operators of d that is a solution to i. Determining whether a domain model solves an instance is PSPACE-Complete (Erol, Nau, and Subrahmanian 1992). We say that d is *solubility-preserving* for instance i if whenever i is solvable in the original domain D, it is solvable in d. Formally, a binary function $sp(i, d) = 1$ if d is solubility-preserving for i, and $sp(i, d) = 0$ otherwise. An example of a solubility-preserving domain model is the original domain augmented with a sound set of macro operators. An example of a domain model that is *not* solubility-preserving for a non-trivial instance is the domain with no operators. The *coverage ratio* of a domain model d, γ_d, is the probability that d is solubility-preserving for an instance i picked at random from the intended distribution. In general, it is hard to find the coverage though it can be estimated empirically given a set of problem instances.

Approach

Our approach aims to learn to remodel planning domains by adding macros and removing basic operators while maintaining a probability of solubility-preservation of the learned domain models in an instance-specific context and in a domain-level context.

Figure 1 presents a schematic diagram of the system. Given an original STRIPS (Fikes and Nilsson 1971) domain and a set of candidate macro operators, offline we enumerate all subsets of macros and selected original operators from the domain, discarding obviously non-solubility-preserving sets such as the empty set. We construct a new domain model with the new set of operators. Given a number of training instances, we evaluate each domain model by running the planner on all instances with the model and registering the results and the run-time. In this way we (1) search for the domain model that most improves the speed of the planner for each instance and (2) estimate the probability of solubility preservation of each domain model. Finally, we use a learning tool to construct a prediction model to choose, based on the features of a problem instance, the best domain model for an instance.

Online, the predictor is consulted: based on the incoming problem instance's features, it suggests a domain model to be used to solve the instance. This approach is applicable to any planner and domain, although a separate predictor must be learned for every planner-domain pair.

Approximating Coverage

While it is hard to prove solubility preservation of the domain models, it is important to maintain an acceptable level of *empirical* coverage, given that the system will be applied to new instances. To measure coverage of a domain model d, we define $\widehat{\gamma_d^I}$ to be the ratio of the number of training instances I that d solved to the total number of solvable training instances. We maintain a lower bound on the coverage with a minimum coverage ratio constant γ.

The coverage ratio term defined in previous section cannot be used directly in here for two reasons: it requires sampling to be measured, and it is unclear how to measure it under the use of timeouts. We can approximate the coverage ratio γ_d for a domain model d by sampling instances from the intended distribution, represented by the set of training instances, and finding whether d is solubility-preserving on these instances. However, this requires knowing whether each sampled instance is solvable or not, which is a PSPACE-complete problem. Therefore, we must approximate γ_d by taking the timeout of the planner into account while sampling.

We say that an instance i is c-solvable in a domain model d by planner P if running P with i and d returns a valid solution plan to i within time c. If P halts normally and returns no plan within time c, then i is *not* c-solvable by d on P.

A domain model d, which is constructed by adding macros from a set of sound macros S, and removing original operators from the original domain D, is c-solubility-preserving for i if whenever the planner P returns a plan for i using some domain model d', constructed from D and S, within time c, we find that running P with d and i also returns a solution plan within time c. We define a new function c-sp as follows:

$$c\text{-}sp(P, i, d) = \begin{cases} 1 & \text{if } d \text{ is } c\text{-solubility-preserving for } i. \\ 0 & \text{if } d \text{ is not } c\text{-solubility-preserving.} \end{cases}$$

The coverage ratio is the expected value of the sp function, which indicates whether a domain model is solubility preserving for an instance. To approximate γ_d, we sample a set of instances I from the intended distribution $Dist_D$ and find the number instances on which d is solubility-preserving. However, it may be impossible to tell whether d is solubility-preserving on some instances. For example, instances on which all domain models timeout cannot be used in this calculation, since their c-solubility-preserving status, the practical form of solubility-preserving, is undefined. Therefore, we filter the sampled instances to include only those for which we surely know that d's c-solubility-preserving value is defined. We do that by introducing a subset we call the accepted sampling subset of I.

The Accepted Sampling Subset I_d^* of I with respect to d is the set of instances $i \in I$ for which the c-$sp(P, i, d)$ is defined. The value of c-sp is defined only when d did not timeout on the instance, and either there is a domain model that c-solves the instance or a macro-only domain model did not timeout on the instance.

The c-coverage ratio, $\widehat{\gamma}_d^I$, of d using I is the proportion of instances from the accepted sampling subset of I, I_d^*, on

which d is c-solubility-preserving to the total instances of I_d^*. More formally:

$$\begin{aligned}\widehat{\gamma}_d^I &= \widehat{E}_{i\in I_d^*}[c\text{-}sp(P,i,d)] \\ &= \frac{\Sigma_{i\in I_d^*} c\text{-sp}(P,i,d)}{|I_d^*|}\end{aligned}$$

For example, when d is the original domain augmented with macros, we know that $\widehat{\gamma}_d^I = 1$ if d can c-solve one instance of I. Notice that $\widehat{\gamma}_d^I$ is not defined if $|I_d^*| = 0$. This can happen when all macro-addition domain models time-out and none of the other domain models returns a solution plan within time c. In this case, there is no domain model that can be used to know whether the instance is really unsolvable, since domain models that remove operators are not solubility-preserving in general.

Macro Source

Any source of macros can produce candidate macros. In our experiments, we use three sources: the chunking phase of Wizard (Newton 2008), the CA-ED version of Macro-FF (Botea et al. 2005), and manually constructed macros. We discuss these sources in detail in the Experiments section.

Training

Given a STRIPS domain D with a set of operators O, a set of macro operators S, a planner P, a set of training instances I_{train}, and the minimum coverage ratio constant, γ, the offline training phase works as follows:

1. We run the operator selector to choose the operators to be removed. If we know that the removal of an individual operator o will increase the number of unsolvable instances above the acceptable level specified by γ, we do not choose it. This step is analogous to the step in which a macro operator tool filters out the less useful individual macros. The operator selector performs the following steps:
   (a) Given a set of small problem instances I_{sel}, the planner is run on each instance in I_{sel} with two models: the domain that contains all initial macro operators S and original operators, d_{all}, and the model, d_o, which is the original domain augmented with the initial macros S, but with one basic operator o removed.
   (b) We calculate d_o's average c-coverage ratio $\widehat{\gamma}_{d_o}^{I_{sel}}$, which is the proportion of instances solved correctly without o divided by the total number of correctly solved instances. We know the correctness of the planner results under d_o by comparing it to the planner results under d_{all}, which is solubility-preserving.
   Here, we have two models to look at: d_{all} and d_o. By looking at the solubility of the instances of I_{sel}^* using both d_{all} and d_o, we can identify the accepted sampling instances I_{sel}^* with respect to d_o as described in the previous subsection. So:
   $\widehat{\gamma}_{d_o}^{I_{sel}} = \frac{\Sigma_{i\in I_{sel}^*} c\text{-}sp(P,i,d_o)}{|I_{sel}^*|}$, where I_{sel}^* is the accepted sampling subset with respect to d_o.
   (c) If $\widehat{\gamma}_{d_o}^{I_{sel}} < \gamma$, we must *not* choose operator o for removal, since we know that any domain model that does not contain that operator is likely to be solubility-preserving only on a proportion of instances less than γ. If $\widehat{\gamma}_{d_o}^{I_{sel}} \geq \gamma$, then we can add o to the set of removable operators Q that is initially empty.
   (d) We repeat the steps (a),(b) and (c) for all domain operators, or until $|Q| = q$, where q is a constant corresponding to the maximum number of operators we allow to be removed. To break ties as a result of q, since more than q operators may be removable, we add to Q the operators o whose corresponding domain models, d_o, have the smallest average run-time on the instances of I_{sel}.
2. We generate all possible domain models that result from adding macros and/or removing operators in Q.
3. We exhaustively run all training instances I_{train} on these domain models and register the results.
4. We *accept* only domain models that solve at least a proportion γ of the training instances correctly, and we discard the other domain models. So:
   $accept = \{d|\widehat{\gamma}_d^{I_{train}} \geq \gamma\}$
   We know that some domain models should be accepted unless all instances time out. For example, the original domain model D (with no macros added or operators removed) should be accepted because $\widehat{\gamma}_D^{I_{train}} = 1$, and all domain models that only add macros should be accepted.
5. Finally, we use the run-times of the accepted domain models to train the *Direct* predictor as in (Alhossaini and Beck 2012). The Direct predictor is built by training a classification model. We associate the problem instance's feature with the accepted domain model that has the smallest run-time on that instance. Then, (features, domain model) pairs are passed to a classification-based learner to produce the predictor. This predictor directly predicts the appropriate domain model for a problem instance based on the given problem instance's features. As the number of domains to choose from is $O(2^{(|Q|+|S|)})$, we restrict the classification to the top r domain models whose removal has the largest effect on the run-time of the best achievable instance-specific predictor based on the training instance.

Features and Learning

To identify features, we used a modified version of an existing feature selection algorithm (Yoon, Fern, and Givan 2008). The algorithm is based on finding features of the problem instance from classes defined using a taxonomic syntax language. A taxonomic syntax language represents general features for arbitrary planning domains and problem instances. The classes correspond to sets of objects that fill particular roles in domain predicates. In our feature selection algorithm, the classes are constructed beginning from level 0, where basic concept classes, such are the 'type' class in STRIPS language, are built. Classes at subsequent levels are recursively constructed from classes at previous levels using language connectors, including class intersection and complement. The features are the number of objects that belong

to each class. We select the features that have the highest coefficient of determination, R^2, value with respect to the run-time of some selected domain model. In statistics, the R^2-value provides a measure of how well the observed outcomes can be predicted by the prediction model (Steel and Torrie 1960). To make the process manageable, we measure R^2-value on domain models where either at most one macro is added to the original domain or at most one operator is removed from the macros-augmented domain. For more details, please see (Alhossaini 2013)

In addition to the taxonomic syntax language features, we also use two simple kinds of features: the number of initial state and goal facts, and some manually extracted, domain-dependent features, only in one domain (freecell). Again, please see (Alhossaini 2013) for complete details.

For learning, we use a non-linear classification algorithm: SVM-SMO (Platt 1999) with quadratic kernel to train the Direct predictor.

Experiments

In this section, we assess the performance of our approaches against existing macro learning tools as well as against other reasonable or perfect knowledge approaches.

Settings

The experiment was conducted in a 30-node cluster, where each node consists of two Dual Core AMD 270 CPUs, 4 GB of main memory, and 80 GB of disk space. We used the Ruby scripting language to write the code. We registered the average run-time of the FF planner (Hoffmann and Nebel 2001) in three experiments, each using a different source of macros (see below). We limit the size of the initial macro set to 5 and set $\gamma = 0.98$ and $q = 3$.

Macro sources

We split our experiments over three different macro sources, using one source for each experiment. We use Wizard's chunking phase (Newton 2008; Newton et al. 2008), Macro-FF's CA-ED version (Botea et al. 2005), and some manually constructed macros.

We used the version of Wizard that was used in the IPC. To get the initial set of macros, we ran the chunking phase of Wizard only to get a set of individually useful macros. We compare Wizard's performance to our predictors by running Wizard's bunching phase using the resulting macros to find a useful subset of these macros. As parameters to the genetic algorithm, we used five epochs, and a population of size 9. We changed the default macro utility threshold in some domains to get macros in all domains

We used the CA-ED version of Macro-FF. CA-ED is a system that extracts macro operators, adds them to the domain, and uses the new domain with the FF planner to solve future instances.

Planning Domains

We used six benchmark domains from the International Planning Competition (IPC). We chose the domains from a spectrum of domains described in (Hoffmann 2001). The domains are chosen to include those with plateaux (logistics), local minima (blocksworld), and dead-ends (pipesworld, mystery, mprime, and freecell). We did not run all macro sources on all domains, since some domains did not work with the macro source.

Problem Instances

All problem instances were generated using a problem generator as we needed a large number of instances for the training.[1] The parameters used to generate the new problems of each domain are shown in Table 1. Details of the parameter meanings can be found in the corresponding generators.

Domain	parameters
logistics	$a \in [1,3], c \in [4,6], s \in [4,6], p \in [50,62]$
blocksworld	$n \in [2,50]$
pipesworld-nt	$p \in [2,4], b \in [3,5], g \in [1,2]$
mprime	$l \in [5,5], f \in [30,30], s \in [1,2], v \in [1,2], c \in [5,8]$
mystery	$l \in [5,5], f \in [30,30], s \in [1,2], v \in [1,2], c \in [5,8]$
freecell	$f \in [4,4], c \in [8,8], s \in [4,4], l \in [13,13]$

Table 1: The parameters for the training and test instances.

Note that the parameters used to generate the instances for all domains in the different experiments (FF-WIZARD, FF-MACROFF, and FF-MANUAL) are different. So, for example, the blocksworld domain training and test instances have different sizes for each experiment. The reason of such difference is that the macros used in each experiment have different performance using FF. If we use the same set of parameters in all experiments we may find that the instances either timeout or are solved very quickly in some experiments, which makes the comparison harder. We chose the parameters that generate random hard instances, but not so hard that no instance can be solved.

Selected Removable Operators

Using the operator selector described earlier, the removable operators are described in Table 2.

Timeout Handling

In every run of a problem instance, we set the cut-off time to one hour. The run-time of timed out instances is registered one hour and the test instance for which all domain models timed out is not considered for the evaluation, since it does not differentiate among the domain models.

Models to Compare

We compare the mean run-time of seven common domain models and domain model predictors:

ORIG is the original unmodified domain.

BOA-M is the best accepted domain model on average on the training instances when allowing only macro addition.

[1] All generators were from http://www.loria.fr/∼hoffmanj/ff-domains.html, except for pipesworld, which is available at http://www.cs.toronto.edu/∼maher/ pipesworld_nt_gen/.

Domain	Selected Operators (Q)
FF-WIZARD	
logistics	{load-truck, unload-truck, unload-airplane}
blocksworld	{pickup, stack, unstack}
pipesworld-nt	{}
mystery-5	{}
freecell	{sendtofree, sendtofree-b, newcolfromfreecell}
FF-MACROFF	
blocksworld	{pickup, stack, unstack}
pipesworld-nt	{push-start, push-end}
freecell	{sendtohome}
FF-MANUAL	
logistics	{load-truck, unload-truck, unload-airplane}
blocksworld	{pickup, putdown, unstack}
pipesworld-nt	{push-start, pop-start, pop-end}
mprime-5	{move, donate}
mystery-5	{move, load, unload}
freecell	{sendtofree, sendtofree-b, colfromfreecell}

Table 2: The operators selected for potential removal.

BOA is the best accepted domain model on average on the training instances with both macro addition and operator removal.

DIR is the direct predictor using exhaustive evaluation of all accepted domain models on the training instances. We set $r = 16$. The time required to measure the problem instance features and to consult the predictor is negligible and so not included in DIR's mean run-time.

PERF is the imaginary perfect predictor that knows in advance the best accepted domain model for every *test* instance. It is based on an exhaustive evaluation of all domain models on the test instances.

WIZ and MFF are the macro-only domain models chosen, respectively, by Wizard and Macro-FF.

Results

We used paired t-test (Cohen 1995) with $p \leq 0.05$ to test for statistical significance. Table 3 shows that the performance of the Direct predictor is significantly better than that of the domain-level remodelling using macro addition only (BOA-M) and using both macro addition and operator removal (BOA). In 9 out of 14 combinations, we found DIR significantly faster than BOA-M, while not being significantly slower in any case. Compared to BOA, DIR was significantly faster in 5 out of 14 cases and was not significantly slower in any case. Compared to the existing macro learning tools, the Direct predictor was never significantly slower while significantly out-performing Wizard in 3 of 5 domains and Macro-FF in 2 of 3 domains.

The perfect predictor (PERF) was significantly faster than all macro-only domain models, including BOA-M, WIZ, and MFF, in all cases, and significantly faster than BOA in 13 out of 14 cases. The run-times of PERF represent an upper bound on performance that can be achieved in our experiments. We observe that in while DIR is close in performance to PERF for low run-time instances, there tends to be a substantial separation on other instances. Narrowing this performance gap is a key challenge for future work.

In terms of coverage, we found that DIR solved all of the 3685 solvable test instances correctly, while BOA incorrectly returned 'no plan was found' in 5 instances where there existed a valid plan in the original domain. Both DIR and BOA maintained a high c-coverage on the test instances.

Discussion

The results demonstrate that remodelling with macro addition and operator removal can achieve a significant planning speed-up over macro-only remodelling methods and the original domains. They also show that the instance-specific remodelling can achieve a significant planning speed-up as compared to domain-level remodelling. All these results are achieved while maintaining high c-coverage ratios.

Feature Selection and Learning

The key aspect of learning instance specific macros is to find a relation between instance features and macro performance. While our results demonstrate that the learning was successful in a number of cases, we would like to further understand the nature of such relations embodied in the predictors.

In the FF-MANUAL/mprime combination, DIR chose from only three sets of operators: $O_1 = \{$move, load, unload, load-move-unload, donate-move$\}$, $O_2 = \{$move, load, unload, donate$\}$, and $O_3 = \{$move, load, unload, donate, load-move-unload$\}$. We also found that DIR mainly used two features to select from these operators: the capacity of the truck, s, and the number of trucks, t.

In mprime, the macro `load-move-unload` works very well. Intuitively, it is a short-cut that moves a package between locations in one step rather than three. However, when the truck can carry more than one package, `load-move-unload` will tend to result in trucks travelling partially empty and then returning to load the next package. Furthermore, the delete relaxation is less informative, as the truck can be in both its original and new location simultaneously, meaning that the heuristic does not realize that the return trip is necessary. Another macro, `donate-move`, was also found to be useful. If there is no fuel in location 'A' for a truck at 'A' to move, the operator `donate` can transfer a unit of fuel from a location 'B' to 'A'. The macro `donate-move` performs the donating and the moving in one step. With only one truck, it is reasonable that the donated fuel should often be used immediately to move the truck. However, when more trucks are available, donated fuel may be used by any truck in the future, which makes forcing one truck to use it immediately less likely to be a good idea.

Table 4 presents detailed results in the mprime domain. The columns labelled "# best" show that the feature s is highly correlated to the performance of macro sets containing the `load-move-unload` macro. When $s = 2$, using the original domain, O_2, without `load-move-unload` is the best choice for 65% of the test instances. When $s = 1$, including `load-move-unload`, O_1 or O_3, is the best choice for about 72% of the test instances. However, when

Domain	ORIG	BOA-M	WIZ / MFF	BOA	DIR	PERF
FF-WIZARD						
Logistics (320)	12.44 (0)	10.17 (0)	12.66 (0)	$^{+\#}$9.35 (0)	$^{*+\#}$8.89 (0)	$^{*+\#}$6.63 (0)
Blocksworld (225)	1270.86 (0)	0.52 (0)	97.82 (0)	$^{+\#}$0.05 (0)	$^{+\#}$0.06 (0)	$^{+\#}$0.04 (0)
Pipesworld-nt (213)	163.51 (0)	167.47 (0)	152.79 (0)	167.47 (0)	177.82 (0)	$^{*+\#}$101.25 (0)
Mystery-5 (216)	155.64 (0)	155.64 (0)	143.98 (0)	155.64 (0)	132.45 (0)	$^{*+\#}$20.26 (0)
Freecell-A (199)	513.21 (0)	359.14 (0)	568.37 (0)	$^{\#}$234.75 (0)	$^{+\#}$118.09 (0)	$^{*+\#}$10.03 (0)
FF-MACROFF						
Blocksworld (250)	1267.8 (0)	2.02 (0)	3.04 (0)	$^{+\#}$0.07 (0)	$^{+\#}$0.07 (0)	$^{*+\#}$0.04 (0)
Pipesworld-nt (289)	255.29 (0)	255.29 (0)	296.34 (0)	336.49 (0)	255.29 (0)	$^{*+\#}$10.32 (0)
Freecell-A (298)	477.01 (0)	302.67 (0)	302.67 (0)	302.67 (0)	$^{*+\#}$206.69 (0)	$^{*+\#}$45.84 (0)
FF-MANUAL						
Logistics (192)	3.70 (0)	3.70 (0)	N/A	3.57 (0)	*3.51 (0)	$^{*+}$3.13 (0)
Blocksworld (300)	1055.03 (0)	3.52 (0)	N/A	$^{+}$0.08 (0)	$^{*+}$0.05 (0)	$^{*+}$0.04 (0)
Pipesworld-nt (356)	347.65 (0)	244.90 (0)	N/A	244.90 (0)	316.12 (0)	$^{*+}$12.45 (0)
Mprime-5 (240)	590.38 (0)	7.28 (0)	N/A	7.28 (0)	$^{*+}$2.12 (0)	$^{*+}$0.23 (0)
Mystery-5 (287)	232.33 (0)	172.93 (0)	N/A	134.92 (4)	$^{+}$82.21 (0)	$^{*+}$1.31 (0)
Freecell-A (300)	438.25 (0)	415.91 (0)	N/A	$^{+}$164.62 (1)	$^{+}$211.67 (0)	$^{*+}$6.22 (0)

Table 3: Mean run-times in seconds followed by the number of incorrectly solved test instances (in parentheses) for the domain models using FF and three macro sources. The number of tested instances is shown in parentheses with each domain's name. The asterisk (*), plus (+), and pound (#) mean that the model was significantly faster than BOA, BOA-M, and (WIZ or MFF), respectively, with $p \leq 0.05$.

operators set	# best			# correct		
	$s=1$		$s=2$	$s=1$		$s=2$
	$t=1$	$t=2$		$t=1$	$t=2$	
O_1	13	10	3	13	0	0
O_2	0	7	78	0	0	78
O_3	30	33	1	0	33	0
other sets	17	10	38	0	0	0

Table 4: Relation between the features (s and t), the operator sets, and the best possible choice of the DIR predictor in the mprime domain. Columns 2–4 are the number of instances on which the corresponding operator set was best. Columns 5–7 are the number of instances on which DIR correctly chose that corresponding best operator set.

$s = 1$ the feature t is somewhat correlated to the choice of O_1 and O_3. When $s = 1$ and $t = 1$, using the original domain's operators with the macro `load-move-unload` (O_1) is the best choice for 22% of the test instances. When $s = 1$ and $t = 2$, using the original domain's operators without `donate` and with `load-move-unload` and `donate-move` (O_3) is the best choice for 55% of the instances. Given that finding a small number of features that can cluster the test instances is hard, and that there are 16 sets to chose from, we may consider these features informative.

The columns labelled "# correct" show the number of instances on which DIR found the best macro set choice. The Direct predictor correctly detected the correlation between s and the `load-move-unload` macro. When $s = 2$ and O_2 is the best set to choose, DIR was 100% correct in choosing O_2. When $s = 1$, $t = 1$ and O_1 is the best set, and when $s = 1$, $t = 2$ and O_3 is the best, DIR was correct 100% of the time in both cases. This result shows a high accuracy of the predictor on the set O_2. Although DIR was less accurate when the best subset was O_1 or O_3, it was able to use the less informative feature, t, to often predict the best sets.

The quality of the selected features and the high accuracy of DIR can explain why DIR outperforms BOA by at least twofold. While the average run-times of domains constructed from the operators of O_1, O_2, and O_3 are 56.01, 590.38, and 107.97 seconds, respectively, BOA's run-time is 7.11 seconds and DIR's run-time was 2.12 seconds. This high performance of DIR is due to the results shown in Table 4: (1) the feature selector found a useful features, and (2) DIR used these features to make accurate predictions. In summary, in 72.91% of the instances, O_1, O_2 or O_3 was the perfect choice. From these instances, DIR correctly chose the perfect set in 70.86% of the cases. In instances where DIR did not choose the perfect set, it chose sets that are strictly faster than BOA in 97.41% of the cases.

Solubility Preservation

While our approach does not guarantee a minimum coverage ratio, we are able to find empirically reliable remodelled domains by accepting only those with average ratios $\geq \gamma$ in training. The results show that the new domain models (e.g.,

BOA) have acceptable c-coverage ratios ($\geq \gamma$) in all planning domains *on the test instances*. Empirically, the direct predictor had a c-coverage ratios of 1 in all domains.

Faster Offline Evaluations

One drawback of the exhaustive evaluation approach is that the number of domain models to evaluate is exponential in the number of macros and operators. Even though the evaluation is achieved offline, and hence the planner can be run in parallel, it may take a long time, since there are many domain models. The median total evaluation time for all domains was 963.18 hours, while the minimum total evaluation time was in the FF-MANUAL/logistics domain with 22.09 hours and the maximum time was in the FF-MANUAL/blocksworld domain with 5710.36 hours.

In this paper, we choose a constant number, $q = 3$, of operators to remove, while limiting the number of macros to less than 6. Using such restrictions limits the number of models, but may also limit the scalability of our approach even if this computation is offline.

We believe that the parameter tuning approach to domain remodelling (Alhossaini and Beck 2012) is a promising way to address this issue. Recasting the domain model components (the macros and the operators) as domain parameters that can be optimized using a parameter tuner can make the approach more practical.

Future Work

Although the feature selection method we used yields good performance, we believe that more work should be done to find a better feature selector. Too many generic features are generated; the features need to be more correlated to the models performance, and their number should to be reduced.

Finding useful features automatically for every domain is challenging, because there are many possible features and different domain models, even though we can sometimes find useful features easily in some domains. Existing work may provide a starting point. In (Roberts et al. 2008), for example, structures such as causal graph were used to predict the performance of planners on problem instances. We may be able to exploit this connection in reverse to use problem instance characteristics to suggest domain remodelling steps via the modifications of such structures. In (Chrpa 2010), some operators are restricted to be only applicable in the initial state (or the state before the goal) if they can be linked to the problem instance's initial state (goal) features. It may be possible to use subsequent states resulting from the application of such a restricted operator in the initial state (or the regression of the goal state using these operators) as an additional source of features in the taxonomic syntax approach.

Conclusions

In this paper, we demonstrated the following:

1. Remodelling by both macro addition and operator removal often significantly improves the planning speed in a state-of-the-art planner, as compared to macro addition only.
2. Instance-specific remodelling using machine learning can lead to a significant improvement in the performance over the domain-level remodelling with macro addition only or with both macro addition and operator removal.
3. The lack of solubility preservation that accompanies the operator removal can be empirically controlled in learning new domain models.

Acknowledgement

This research is supported by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Research Fund, Microway Inc., and King Saud University.

References

Alhossaini, M., and Beck, J. C. 2012. Macro learning in planning as parameter configuration. In *Proceedings of the 25th Canadian Conference on Artificial Intelligence*, 13–24. Springer.

Alhossaini, M. 2013. *Remodeling Planning Domains Using Macro Operators and Machine Learning*. Ph.D. Dissertation, University of Toronto, forthcoming.

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.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, 1659–1664. Morgan Kaufmann Publishers Inc.

Chrpa, L. 2010. Combining learning techniques for classical planning: Macro-operators and entanglements. In *Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, 79–86. IEEE Computer Society.

Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass.

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

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To Max or Not to Max: Online Learning for Speeding Up Optimal Planning. *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-2010)*.

Erol, K.; Nau, D.; and Subrahmanian, V. 1992. On the complexity of domain-independent planning. *Proc. AAAI-92* 381–386.

Ferrara, A.; Liberatore, P.; and Schaerf, M. 2005. The Complexity of Action Redundancy. *AI* IA 2005: Advances in Artificial Intelligence* 1–12.

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.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann; Elsevier Science.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *Proceedings of Artificial Intelligence Planning Systems (AIPS)*, 150–158.

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.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36(1):267–306.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring Irrelevant Facts and Operators in Plan Generation. In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, 350. Springer-Verlag.

Newton, M.; Levine, J.; Fox, M.; and Long, D. 2008. Wizard: Compiled Macro-Actions for Planner-Domain Pairs. *Booklet for the 6th International Planning Competition Learning Track*.

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

Platt, J. 1999. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods*, 185–208. MIT Press.

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.

Steel, R., and Torrie, J. 1960. *Principles and procedures of statistics: with special reference to the biological sciences*. McGraw-Hill Companies.

Vrakas, D.; Tsoumakas, G.; Bassiliades, N.; and Vlahavas, I. 2003. Learning rules for Adaptive Planning. *Proceedings of the 13th International Conference on Automated Planning and Scheduling, Trento, Italy* 82–91.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *The Journal of Machine Learning Research* 9:683–718.