Macro Learning in Planning as Parameter Configuration

Maher Alhossaini[1] and J. Christopher Beck[2]

[1] Department of Computer Science, University of Toronto
[2] Department of Mechanical & Industrial Engineering, University of Toronto
maher@cs.utoronto.ca, jcb@mie.utoronto.ca

Abstract. In AI planning, macro learning is the task of finding subsequences of operators that can be added to the planning domain to improve planner performance. Typically, a single set is added to the domain for all problem instances. A number of techniques have been developed to generate such a macro set based on offline analysis of problem instances. We build on recent work on instance-specific and fixed-set macros, and recast the macro generation problem as parameter configuration: the macros in a domain are viewed as parameters of the planning problem. We then apply an existing parameter configuration system to reconfigure a domain either once or per problem instance. Our empirical results demonstrate that our approach outperforms existing macro acquisition and filtering tools. For instance-specific macros, our approach almost always achieves equal or better performance than a complete evaluation approach, while often being an order of magnitude faster offline.

Keywords: Planning, Macro Learning, Parameter Configuration, Machine Learning

1 Introduction

Classical AI planning is the problem of finding a sequence of operations that transforms the world from an initial state to a goal state. As it is assumed that a planning domain has many problem instances, differing for example on the initial and/or goal state, it is advantageous to add domain knowledge or control rules to improve planning for multiple instances [6]. One form of domain knowledge is a *macro operator*, a sequence of original operators added as a single unit to a domain [11]. The intuition is that macros are solutions to frequently occurring subproblems and so adding them as an operator allows the planner to re-use rather than re-discover a sub-plan.

Recently, we introduced *instance-specific* macros [1]. Rather than adding one set of macros to a domain to use for all instances, we learn a predictor that maps problem instance features to macro set performance. Online, the features of a new problem instance are measured and the predictor is used to choose the macro set that is added to the domain for that instance.

The core idea of this paper is to recast macro learning as parameter configuration. The latter problem is typically formulated from an algorithmic perspective:

given a complex algorithm with a variety of tuning parameters, find parameter values that result in the best performance of the solver on a given set of problem instances. Systems such as ParamILS and F-race have resulted in orders of magnitude increases in performance for mixed integer program solvers, SAT solvers, and local search algorithms [2, 9]. We propose an analogy between macro learning and parameter configuration. While the latter directly changes the algorithm behaviour, the former changes the problem instance. Just as algorithm behaviour can be manipulated via parameter values to better match a problem instance, we conjecture that a problem instance can be remodeled to make it more appropriate for a given algorithm. We investigate this analogy by developing a fixed and instance-specific macro learning system that uses the ParamILS [9] parameter configurator.

We also address a weakness in the previous instance-specific macro work: the requirement to solve an exponential number of planning problems, albeit offline. We show that a parameter configuration approach can achieve the same online performance without the need for such substantial offline computation.

In the next section, we present background on fixed and instance-specific macro learning and parameter configuration. We then detail the approach used here to formulate macro learning as a parameter configuration problem. Section 4 details our experimental set-up and empirical results. We compare our work against existing macro learning systems and demonstrate that our technique can find better macros in the fixed-macro-set scheme and can significantly reduce the preprocessing time in instance-specific learning while choosing macros that are often as efficient. Section 5 discusses our results and finally we conclude.

2 Background

2.1 Classical Planning

In classical planning, a *domain* is set of operators, predicate symbols, and objects. The operators and predicates can be grounded into actions and fluents (respectively) using variable assignment. A *problem instance* is represented by an initial state and a goal partial state, each represented by a set of fluents. An action transforms a state by changing the values of the fluents. A solution *plan* is a sequence of actions that transforms the initial state into a goal state.

2.2 Learning Macros

The standard, fixed-macro approach is to add macro operators to the domain and then solve all subsequent problem instances with the augmented domain. As a consequence, research has focused on finding one set of macros that improves the *average* performance of a planner on a domain. As noted above, we introduced the idea of *instance-specific* macro learning that chooses macros based on the features of a problem instance [1]. We review both types of macro learning here.

A macro is a sequence of operators that are aggregated and added to a domain definition as a new operator. Macros speed-up planners since they can act as shortcuts to deeper states in the search tree [3, 5, 11]. However, adding macros can harm performance by increasing the branching factor.

Macro-FF [3] is a macro-learning system that identifies *abstract components*, sets of related objects that are unrelated to other objects in the domain. Macros are generated from abstract components and then ranked and filtered by running the planner plus macro on a set of training instances. In the CA-ED version of Macro-FF, the chosen macro operators are then added to the domain definition. Macro-FF demonstrated a significant improvement in the planning speed in a number of benchmark domains that contain abstract components.

The Wizard macro acquisition tool [11] uses a genetic algorithm to learn macro operators from randomly generated problems. Wizard lifts macros from solutions to a set of small problem instances and then iteratively modifies them using genetic operators in its *chunking* phase. The macros are evaluated, based on planning speed, on a set of larger problem instances. The *bunching* phase then uses a second genetic algorithm to find the best sets of the macros from the chunking phase. The macros suggested by Wizard were shown to lead to a significant improvement in planning speed [10].

In instance-specific macro learning, Alhossaini & Beck [1] follow an *exhaustive* approach, learning a mapping between features of a problem instance and performance of macro sets. Offline, based on an initial list of macros from an external source, the system solves each learning instance with each macro subset. The run-time data from these runs form the basis for learning the mapping. The predictor is then used online: the features of a new problem instance are measured and the predictor is used to select the macro subset expected to perform best. The primary components of the system are as follows:

- Macro sources: The system assumes an initial set of macros from which it forms and evaluates all subsets. Any macro sources can be used. See Section 4.1 for details of the sources used here.
- Feature extraction: The problem instance features are based on a detailed taxonomic syntax language [14] and some simple statistics of the initial and goal states (e.g., the number of facts).
- Learning approach: A *direct predictor* attempts to predict the macro subset that will be fastest for a given problem instance. The original problem domain, target planner, training set of problem instances, and the initial set of n macros are provided as input. Each of the $k = 2^n$ subsets is run on each training instance. The feature measurements of each instance and run-time of each macro subset on that instance form the basis for learning, using WEKA's SMO algorithm [13, 12] to map the problem instance features to predicted fastest macro subset.

Experimental results demonstrate that learning to choose instance-specific macro subsets can significantly outperform the standard fixed-macro-set approach in a number of domains [1].

2.3 Parameter Configuration

State-of-the-art solvers are often designed with a number of parameters that can be used to modify the solver's behaviour. While such parametrization significantly adds to the solver's customizability, the right parameter values must be found to maximize performance. Automated parameter tuning has made significant strides over the past few years and is often able to find parameters that achieve performance substantially better than can be achieved by an expert or, indeed, the original designer of the solver [9].

ParamILS is a general-purpose algorithm tuner designed to deal with a very large space of numerical and categorical parameters [9]. The input to ParamILS is the target solver, the parameters to be optimized, and the set of training problems. The output is a parameter configuration designed to maximize performance on problem instances similar to the training set. ParamILS uses a number of techniques to speed up the search through the reduction in the number of training instances and run-time needed to distinguish among different configurations. ParamILS configurations were able to outperform the default ones of a number of well-known AI algorithms and tools [9].

3 Macro Learning as Parameter Configuration

The overall goal of macro learning is to improve planning performance by adding a set of useful macros to a domain, either once, in the usual fixed-macro approach, or for each problem instance, in instance-specific macro learning. Our hypothesis is that we can use a parameter configuration tool to configure a domain to better suit either a particular planner or a planner/problem instance pair.

Fixed Macro Set Learning To learn a fixed macro set, we use ParamILS in exactly the way it is used for parameter configuration. The input is a planner, a domain definition, an initial ordered set of macros, S, and a set of training problems. S is encoded as $|S|$ binary parameters, each one indicating the presence or absence of a macro in the macro set. We represent a macro set as a vector of these indicator variables. For example, if $|S| = 4$, the parameter configuration $(1, 0, 0, 1)$ indicates a macro set using only the first and fourth macros.

We then run ParamILS as a black box to determine which binary vector represents the "parametrization" predicted to lead to the lowest run-time for the planner, domain, and problems instances similar to the training instances. The chosen macro set is added to the domain and the set of testing instances (disjoint from the training set) are then solved with the augmented domain.

Instance-Specific Macro Learning Building on [1], our approach is to generate a *direct* predictor that, given a set of problem instance features, will return a macro set predicted to be best for that instance. Given a planner, domain, an initial set of macros, S, and a set of training instances, I_{train}, we take the following steps:

1. Extract the features of the training instances to get $N \leq |I_{train}|$ different feature settings. A feature setting is an instantiation to each feature variable. Some instances may share the same feature values.
2. For each subset of training instances, I_j, that share the same feature setting, j, use ParamILS as above to find the parameter configuration predicted to minimize the run-time of the planner on I_j. The parameter configuration is the macro subset to be used to solve instance similar to those in I_j.
3. Record the feature setting with the chosen macro subset in the training file.
4. After all training subsets have been processed, input the training file to a supervised learning tool. We use the SMO algorithm with a quadratic kernel from WEKA version 3.4.13 [13] to learn the mapping between feature setting and macro subset.

Note that there are two levels of learning in our instance-specific system: (1) the learning that ParamILS does to identify the best macro set for a given feature setting, and (2) the learning that generalizes the output of the ParamILS runs to map any feature setting to a macro subset.

The predictor is consulted for each test instance. The inputs to the predictor are the measured problem instance features and the output is the macro set that is predicted to perform best with the instance.

4 Experiments

Our experiments are designed to evaluate the planning performance of parameter configuration-based macro learning in both fixed-macro and instance-specific contexts. We compare the mean planning time of a number of domain models on the testing instances using a number of benchmark domains, planners, and macro sources.

4.1 Experimental Set-up

We follow the general experimental set-up of Alhossaini & Beck [1], using the same problem domains, planners, macro sources, and feature selection technique.

Augmented Domains Given a planning domain, a set of macros, and a set of training and testing instances, we compare the following augmented domains:

BOA the *best-on-average* domain model. The domain with the smallest mean run-time based on exhaustively solving all training instances with each macro subsets. BOA represents the best fixed domain model that a standard macro learning approach can provide given the input macros.

BOA-P the *best-on-average* using ParamILS. It is the domain generated by the ParamILS-based fixed-macro-set learning.

DIR the *Direct* instance-specific predictor [1], described above in Section 2.2.

DIR-P the *Direct* instance-specific predictor based on ParamILS (see Section 3).

MIC the domain augmented with the macros that *Macro-FF CA-ED* produced.

ORIG the original unaugmented domain.

PERF the *perfect* predictor. An imaginary instance-specific domain model generated by assuming that for a given *test* instance, the best macro set could be chosen correctly. PERF is the best possible predictor.

PERF-P the *perfect* predictor using ParamILS and the test instances for *both* training and testing. PERF-P does not necessarily suggest the best macro set for each test instance but rather the best macro set chosen by ParamILS run on the test instances with the same feature measurements.

WIZ *Wizard*'s macro set.

Planners We experiment with two planners based on substantially different technology: FF [8] a heuristic forward state-based planner, and VHPOP [15], a partial-order-plan-space planner.

Problem Domains We use benchmark domains from the International Planning Competition. We categorize the domains into *easy* (logistics and blocksworld) and *hard* domains (mprime, mystery, freecell, pipesworld) based on the existence of dead ends and the ability of the $h+$ heuristic to detect them [7].

Training Sets and Test Sets The problems are generated from the problem generators that come with the FF planner,[3] except for pipesworld domain, where the generator was not available and so we wrote one. Table 1 presents the parameter settings and the sizes of the training and test instances for each domain.

Sources of Macros We use three sources of macros: Wizard's chunking phase [10, 11], Macro-FF version CA-ED [3], and the manually constructed macros in Table 2. For Wizard, the default utility threshold was lowered to 150, 100, 100, 100, and 15 for logistics, blocksworld, mystery, pipesworld, and freecell, respectively, to produce macros in all domains. We capture the macros that

[3] http://www.loria.fr/ hoffmanj/ff-domains.html

Domain	parameters	training size	test size
logistics	$a \in [1,3], c \in [4,6], s \in [4,6], p \in [20,52]$	256-480	192-247
blocksworld	$n \in [2,50]$	150-385	97-271
pipesworld-nt	$p \in [2,4], b \in [1,9], g \in [1,3]$	297-420	99-245
mprime	$l \in [5,5], f \in [30,30], s \in [1,2], v \in [1,2], c \in [2,8]$	400-560	143-320
mystery	$l \in [5,5], f \in [30,30], s \in [1,2], v \in [1,2], c \in [2,8]$	400-700	181-296
freecell	$f \in [2,4], c \in [2,8], s \in [2,4], l \in [3,13]$	350-450	150-251

Table 1. The parameters used to generate the instances and the training and test set size ranges. A description of the parameters can be found with each domain generator. For the pipesworld, p, b, g represent the number of pipes, number of extra batches (added to a fixed number of batches that depends on p), and number of goals, respectively. The sizes of the training and testing varied in the experiments; for example, we needed more training instance for the VHPOP planners than FF.

result from Wizard's chunking phase and use them as our initial set of macros. For Macro-FF, the macros were obtained directly from the authors' website.[4]

Feature Selection and Extraction We use the following features sources: (1) a taxonomic syntax language similar to [14]; (2) basic statistics of the problem instance's initial state and goal facts (e.g., the number of initial state and goal facts); and (3) simple, manually extracted, domain-specific features. We use a combination of the features of (1) and (2) in all domains, and we add the features of (3) in the freecell domain.

Timeout and Overhead Handling Offline, in every run of a training instance in the exhaustive condition (i.e., DIR, BOA, and PERF), we set the cut-off CPU time to one hour and use that value as the recorded run-time of timed-out instances. For ParamILS learning, the time to generate a macro set for a given cluster of instances was varied depending on the number of initial macros and the difficulty of the original domain. For most experiments the cut-off time was one or two hours. For easy domains the time was decreased (e.g., the logistics-FF-WIZARD cut-off was 0.25 hours) while for harder domains it was increased (e.g., the mystery-VHPOP-MANUAL cut-off was four hours).

Online, one CPU hour is given to solve each test instance. The run-time of timed-out instances is recorded as one hour and the test instances for which all macro sets timed out are not considered in the evaluation. Instances that were not solved for other reasons (e.g., insufficient memory) are treated as timed-out. The time for feature extraction, running a predictor, and adding a macro to a domain is negligible and so, in practice, the entire hour is used in planning.

Hardware and Software 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.

We use ParamILS version 2.3.2 downloaded directly from the author's website.[5] We used Ruby to write all the code.

[4] http://abotea.rsise.anu.edu.au/index.php?page=macroff.
[5] http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/ on 28 Oct. 2009.

Domain	Macro operators
logistics	unload-airplane-load-truck, load-truck-drive-truck-unload-truck
blocksworld	pickup-stack, unstack-putdown, unstack-stack, unstack-putdown-unstack-putdown
pipesworld-nt	push-start-push-end, pop-start-pop-end, push-unitarypipe-pop-unitarypipe, push-start-push-end-2
mprime	load-move-unload, load-move, move-unload, donate-move
mystery	load-move-unload, load-move, move-unload
freecell	sendtofree-sendtohome, sendtofree-move-colfromfreecell, sendtofree-sendtofree

Table 2. Our manually constructed macros.

4.2 Experiment 1: Fixed Macro Set Learning

In this experiment, we compare the performance of the best fixed-macro-set found by ParamILS to that produced by Wizard and Macro-FF. To achieve this, we evaluate the mean run-time of the following models: ORIG, BOA, BOA-P, MIC, WIZ. Using all benchmark domains, when possible, we conduct three sub-experiments using different planners and macro sources:

1. FF and macros from Wizard's chunking phase.
2. FF and macros from Macro-FF's CA-ED version.
3. VHPOP and macros from Macro-FF's CA-ED version.

In the first experiment we compare against WIZ and latter two against MIC. Our hypothesis is supported if we find evidence that BOA-P is significantly faster than WIZ and MIC and not significantly different from BOA. We use paired t-test with $p \leq 0.05$ [4] to measure significance.

Results Table 3 compares the ParamILS-based fixed-macro method with Wizard. In two out of five domains, BOA-P was found to be significantly faster than WIZ while showing no significant difference in the other three domains. Over all domains, there is no significant difference between BOA and BOA-P.

Table 4 compares our ParamILS-based fixed-macro approach to Macro-FF's macros using the FF and VHPOP planners. In four out of six domain/planner combinations, BOA-P is 1.7 to 20 times faster than MIC. These differences are all significant. In the remaining two domains BOA-P was identical to MIC. In all domains, BOA-P was not significantly different from BOA.

4.3 Experiment 2: Instance-Specific Macro Learning

Using all benchmark domains, when possible, we conduct five sub-experiments:

1. FF and macros from Wizard's chunking phase.
2. FF and macros from Macro-FF CA-ED.
3. FF and manually constructed macros.
4. VHPOP and macros from Macro-FF CA-ED.

Domain	ORIG	BOA	WIZ	BOA-P
logistics	1.20	0.77	1.21	$^{+}$0.77
blocksworld-4ops	1278.00	0.56	100.82	$^{+}$0.56
pipesworld-nt	163.51	167.47	152.79	163.51
mystery-5	155.60	155.60	264.29	264.29
freecell-A	500.91	503.51	536.37	441.65

Table 3. Experiment 1.1: Average run-times (in seconds) of the domain models constructed from chunking phase of Wizard using the FF planners. The plus (+) means that WIZ (BOA-P) was significantly faster than BOA-P (WIZ) with $p \leq 0.05$.

5. VHPOP and manually constructed macros.

Our interest here is to test whether our ParamILS-based instance-specific macro set (DIR-P) is as strong as the exhaustive instance-specific macro set (DIR), while requiring significantly less offline computation. In the experiments where DIR is significantly faster than BOA, we hypothesize that DIR-P will also be faster than BOA. If PERF is not significantly faster than BOA, it is unlikely to see any gain from instance-specific macros due to a ceiling effect. Similarly, if PERF-P is not significantly faster than BOA, there is little for DIR-P to learn in order to be faster than BOA.

Results In Table 5, we show the run-time of the domain models in the five sub-experiments.

DIR vs. DIR-P Online As noted, we would like to show that in combinations of macro source/planner/domain where DIR is significantly faster than BOA, DIR-P is also better than BOA. Overall, of the five combinations where DIR was significantly better than BOA, DIR-P was also significantly better than BOA in four of them. In more detail, the support for our hypothesis is as follows:

– In two sub-experiments, FF-WIZARD and FF-MACROFF, there were no domains such that DIR was significantly better than BOA and so these domains do not address our hypothesis.
– In two sub-experiments, FF-MANUAL and VHPOP-MACROFF, a total of four of the nine planning domains exhibited significant difference between DIR and BOA. On each of these domains, DIR-P was also significantly better than BOA, supporting our hypothesis.
– In VHPOP-MANUAL, there was one planning domain (blocksworld) where DIR was significantly better than BOA, albeit by a small margin. DIR-P was not significantly different from BOA on this domain, failing to support our hypothesis.

Domain	ORIG	BOA	MIC	BOA-P
FF				
blocksworld-4ops	1109.51	56.01	1146.32	$^+$56.01
pipesworld-nt	413.32	413.32	424.90	424.90
freecell-A	371.27	174.79	174.79	174.79
VHPOP				
blocksworld-4ops	1067.45	191.07	533.72	$^+$191.97
pipesworld-nt	294.48	294.48	668.21	$^+$294.48
freecell-B	326.07	326.07	558.36	$^+$326.07

Table 4. Experiments 1.2 and 1.3: Average run-times (in seconds) of the domain models constructed by Macro-FF using the FF planner. See Table 3 for details of the columns and symbols.

Domain	ORIG	Exhaustive				ParamILS			
		BOA	DIR	PERF	Pre (h)	BOA-P	DIR-P	PERF-P	Pre (h)
FF-WIZARD									
logistics	1.20	0.77	0.77	*0.68	0.41	0.77	0.77	0.75	6.76
blocksworld-4ops	1278.00	0.56	22.14	*0.31	645.63	0.56	36.90	*0.38	11.66
pipesworld-nt	163.51	167.47	184.57	*101.25	1416.66	163.51	171.98	134.72	277.00
mystery-5	155.60	155.60	153.93	*40.33	432.67	264.29	191.24	*44.70	135.09
freecell-A	500.91	503.51	476.97	*28.12	772.39	441.65	*349.99	*35.52	145.45
FF-MACROFF									
blocksworld-4ops	1109.51	56.01	56.01	29.78	984.92	56.01	56.01	43.33	159.55
pipesworld-nt	413.32	413.32	425.60	*48.91	120.88	424.90	421.42	*49.09	54.65
freecell-A	371.27	174.79	243.17	*36.14	76.79	174.79	297.91	*36.71	57.73
FF-MANUAL									
logistics	3.80	3.80	3.85	*3.59	209.33	3.80	3.85	3.85	68.10
blocksworld-4ops	1118.18	53.87	53.87	33.69	983.11	53.87	53.86	33.7	58.27
pipesworld-nt	477.4	277.85	440.39	*9.51	1026.23	475.79	472.99	*10.33	100.83
mystery-5	239.48	239.48	*146.00	*39.46	470.10	239.48	*157.81	*71.24	79.98
mprime-5	521.53	8.81	*3.34	*0.29	903.28	83.67	*3.34	*0.30	33.13
freecell-A	296.43	325.21	*194.53	*32.32	442.94	296.43	*191.92	*61.25	106.64
VHPOP-MACROFF									
blocksworld-4ops	1067.45	191.07	191.07	*4.73	1059.14	191.97	191.97	90.70	224.02
pipesworld-nt	294.48	294.48	260.47	*31.41	500.29	294.48	259.86	*51.86	180.33
freecell-B	326.07	326.07	*199.34	*94.31	367.41	326.07	*185.50	*108.50	139.19
VHPOP-MANUAL									
logistics	316.06	316.06	282.26	*163.55	879.34	316.06	*220.91	*170.63	215.34
blocksworld-4ops	896.10	121.26	*121.22	*6.85	1535.37	314.94	173.54	*30.46	114.31
pipesworld-nt	297.86	297.86	297.85	*20.09	4949.29	297.86	673.03	115.80	472.59
mystery-5	433.73	92.36	251.25	*20.67	1211.52	92.36	361.06	48.94	136.87
mprime-5	1048.81	292.49	597.20	*122.08	5991.99	1048.81	790.82	*161.83	738.67
freecell-B	1154.35	214.58	192.34	*47.90	888.02	214.58	277.89	*64.79	144.69

Table 5. Experiment 2: Average run-times (in seconds) of the domain models constructed from different macro sources using different planners. There are five experiments: one for every pair of a planner and a macro source (except VHPOP and Wizard.) The asterisk (*) means that the model was significantly faster than BOA with $p \leq 0.05$. Pre is the preprocessing time measured in hours.

If we compared DIR with DIR-P directly, of the 23 macro source/planning/domain combinations, DIR and DIR-P are statistically indistinguishable in 19 cases, and DIR is significantly better in four cases, all using VHPOP-MANUAL.

DIR vs. DIR-P Offline As expected, the offline learning time for DIR-P is substantially smaller than that of DIR. Over the 23 planner/domain combinations DIR-P has significantly less learning time in 22. The exception is logistics-FF-WIZARD where very short planning times for the training instances led to 16 times faster evaluation for the exhaustive approach. Across all macro source/planner/domain combinations, the median speed-up for DIR-P is 6.5.

5 Discussion and Analysis

We set out to investigate the utility of recasting macro learning as parameter configuration: configuring parameters of the domain rather than of the solver. For learning fixed macro sets, we showed that ParamILS-based macro learning performs as well or better than state-of-the-art macro learning tools: performing better in six domains while showing no significant difference in five. In 10 of these

11 domains the macro set proposed by our system was as good as the best-on-average macro set, representing a reasonable upper bound on the performance of any fixed-macro set approach.

In the context of instance-specific macro learning we compared a predictor that maps problem instance features to macro sets generated by ParamILS to a predictor created by solving an exponential number of problem instances. Of the five macro source/planner/domain combinations where the latter predictor outperformed the best-on-average fixed macro set, the ParamILS-based predictor outperformed the best-on-average in four of them. Comparing the two predictors directly, ParamILS equalled the performance of the exhaustive predictors in 19 of 23 the combinations.

To lend further support to the instance-specific conclusions, we note that PERF-P, the predictor built with ParamILS using the *testing* instances performed as well as PERF on many instances. This result demonstrates that the predictor quality is not reduced by much in moving from the exhaustive approach to the parameter configuration approach.

Interestingly, in two domains: freecell-FF-WIZARD and logistics-VHPOP-MANUAL, we found that the ParamILS-based predictor was better than the best-on-average macro set while the exhaustive predictor was not. We speculate that part of the reason for this is that ParamILS is better able to avoid overfitting because it finds a macro set that performs best on a cluster of problem instances with identical features. However, this cannot be the entire explanation as for logistics-VHPOP-MANUAL, the cluster size was one. Further research is need to understand if this is a real phenomenon and, if so, its source.

In future work, we plan to investigate if parameter configuration can be used for domain remodeling where macros can be added and original operators can be removed. The goal is to find a set of operators and/or macros constructed from these operators that improves the planning speed while maintaining the solubility of the problem instance.

6 Conclusion

In this paper, we presented a novel approach to learning macro operators in AI planning by recasting the macros as parameters of the domain and using an existing parameter configuration tool to identify a good macro set. We evaluated the performance of this approach for traditional fixed-macro learning and for instance-specific macro learning.

In the fixed-macro case, we transform the problem of finding the best subset of macros to parameter configuration where the macro set is represented by a vector of binary parameters. Compared to existing macro learning tools, our approach using the ParamILS tool often finds significantly faster macros sets.

In instance-specific learning, macro sets are chosen for each problem instance based on its features. Offline, our approach clusters training instances based on their features. For each cluster, we then use parameter configuration as in the fixed-macro case to find a strong macro set. The vector of feature measurements and the macro set for each cluster are then input to a standard machine learning algorithm to produce a predictor that maps instance features to macro sets. We found that this approach requires substantially less learning time and produces macros as effective as those produced by an existing instance-specific macro predictor that requires solving with an exponential number of macro subsets.

Taken together, these two results strongly support our hypothesis that macro learning can be fruitfully viewed as parameter configuration.

Acknowledgements 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

1. Alhossaini, M., Beck, J.C.: Learning Instance-Specific Macros. In: ICAPS 2009 Workshop on Planning and Learning (2009)
2. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. Hybrid Metaheuristics pp. 108–122 (2007)
3. Botea, A., Enzenberger, M., Muller, M., Schaeffer, J.: Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. Journal of Artificial Intelligence Research 24, 581–621 (2005)
4. Cohen, P.R.: Empirical Methods for Artificial Intelligence. The MIT Press, Cambridge, Mass. (1995)
5. Coles, A., Smith, A.: Marvin: A heuristic search planner with online macro-action learning. Journal of Artificial Intelligence Research 28(1), 119–156 (2007)
6. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann; Elsevier Science (2004)
7. Hoffmann, J.: Analyzing search topology without running any search: On the connection between causal graphs and h+. The Journal of Artificial Intelligence Research 41, 155–229 (2011)
8. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
9. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36(1), 267–306 (2009)
10. Newton, M.: Wizard: Learning Macro-Actions Comprehensively for Planning. Ph.D. thesis, Department of Computer and Information Science, University of Strathclyde, United Kingdom (November 2008)
11. Newton, M., Levine, J., Fox, M., Long, D.: Learning macro-actions for arbitrary planners and domains. In: Proceedings of the ICAPS (2007)
12. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Advances in kernel methods. pp. 185–208. MIT Press (1999)
13. Witten, I., Frank, E.: Data mining: practical machine learning tools and techniques with Java implementations. ACM SIGMOD Record 31(1), 76–77 (2002)
14. Yoon, S., Fern, A., Givan, R.: Learning control knowledge for forward search planning. The Journal of Machine Learning Research 9, 683–718 (2008)
15. Younes, H., Simmons, R.: VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research 20(1), 405–430 (2003)