Learning Macro-Actions Genetically from Plans

M.A. Hakim Newton, John Levine, Maria Fox, Derek Long
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
e-mail: {newton, johnl, maria, derek}@cis.strath.ac.uk

Abstract

Despite recent progress in planning, many complex domains and even larger problems in simple domains remain hard and challenging. One way to achieve further improvement is to utilise knowledge acquired for the planner from the domain. Macro-actions are a promising means by which to convey such knowledge. A macro-action, or macro in short, is a group of actions selected for application as a single choice. Most existing works on macros utilise knowledge explicitly specific to the planners and the domains. But presumably any particular properties are not likely to be common with different planners or wider range of domains. Therefore, a macro learning system that does not exploit any structural knowledge about planners and domains explicitly is of immense interest. This paper presents an offline system capable of learning macros genetically from plans. Given a planner, a domain, and necessary problems, our system generates macros, lifting from plans of some problems, under guidance from a genetic algorithm. It represents macros like regular actions, evaluates them individually by solving other problems, and suggests the best macro to be added to the domain permanently. Genetic algorithms are automatic learning methods that can learn properties of a system using no explicit knowledge about it. Our system thus does not strive to discover or utilise any structural properties specific to a planner or a domain.

Introduction

Planning is an important research area in Artificial Intelligence. The planning problem is to find a sequence of actions, known as a plan, that takes a given world from a specified initial state to a desired goal state. However, optimal plans that optimise given objective functions are also of concern. There are varieties of problems that require planning research. These include control programs for rovers in planetary exploration, artificial ants, robot navigation systems, block-stacking systems, disaster management, space exploration, military planning, etc.

Planning has achieved significant progress in recent years. The planning competitions held over the last few years (McDermott 2000; Bacchus 2001; Long & Fox 2003; Hoffmann *et al.* 2004) played an important role behind this. In successive competitions, state-of-the-art planners appeared and performed better than their predecessors. Two such recent planners that performed best in the competitions are FF (Fast Forward) (Hoffmann & Nebel 2001) and LPG (Local search for Planning Graphs) (Gerevini & Serina 2002). FF is a deterministic planner that uses a forward chaining heuristic based on a relaxed *graphplan* algorithm. LPG, on the other hand, is stochastic in nature and uses a heuristic inspired by *walksat* in SAT-problems.

Macro-Actions

A *macro-action*, hereinafter *macro* in short, is a group of actions selected for application at one time like a single action. Learning and using macros in planning have been found promising in achieving significant improvement. Macros are used to encode explicit characteristics of particular domains or planners and later exploited to eliminate and reduce search.

Related Works

Macros are not very new in planning research. STRIPS (Fikes, Hart, & Nilsson 1972), an early planning and learning system, solves problems by means-ends analysis and produces macro-operators, called MACROPs, by parameterising the plan wholly. Any subsequences of MACROPs are then used as composite actions to speedup future planning. STRIPS considers every unique subsequence of all previously acquired plans as potential MACROPs; which can quickly lead to an explosion. The REFLECT system (Dawson & Siklóssy 1977) has a preprocessing stage where macro-operators, called BIGOPs, are generated by comparing the post conditions of each action with the preconditions of all possible successor actions, creating a macro whenever they match. This approach severely constrains the possible action sequences and solely focuses on domain characteristics ignoring their impact on a particular planner. MORRIS (Minton 1985) adopts selective measures for STRIPS to learn plan fragments that are used frequently in plans or are helpful in solving difficult subproblems. Macro Problem Solver (MPS) (Korf 1985), which is an extension of General Problem Solver (GPS) (Newell & Simon 1972) to support macros, learns a complete set of macros that totally eliminates the search for a particular goal. MPS achieved success mainly on domains that exhibit operator decomposability and it needs a different set of macros when problem instances scale or goals are different. MACLEARN (Iba 1989) learns macros from sequences of actions that lead the search

to reach a peak from another peak in its heuristic profile. It then represents macros like normal actions, and using static and dynamic filtering, it selects a small set of macros to use in future planning. MARVIN (Coles & Smith 2004) and Macro-FF (Botea *et al.* 2005), both using FF style search algorithm, showed some improvement with macros. Macro-FF learns macros using component level abstraction based on static facts. It also lifts partial-order macros of length two from plans and then ranks them by solving the same problems with macros. MARVIN, on the other hand, learns macros from plans of a reduced version of a given problem after eliminating symmetries. MARVIN saves sequences of actions that lead the search to successfully escape plateaus, and then uses them to escape similar plateaus while solving the original problem. Moreover, macros that achieve heuristically identified subgoals (Hernádvölgyi 2001), show about 44% improvement in the Rubik's Cube domain. Another approach of learning macros automatically by discovering abstraction hierarchies and exploiting domain invariants (Armano, Cherchi, & Vargiu 2005) caused a slightly negative impact on the performance.

Our Contribution

As noted above, most macro learning systems are more focused and specialised to exploiting particular properties of the planners and the domains. But presumably any particular properties are not likely to be common with different planners or wider range of domains. Therefore, a macro learning system that does not exploit any explicit structural knowledge about planners and domains remains unexplored. However, we applied genetic algorithms to evolve macros recently (Newton, Levine, & Fox 2005); further experiments are still being carried out. This paper presents an offline system capable of learning macros genetically from plans. Given a planner, a domain, and necessary problems, our system learns macros from plans under guidance from a genetic algorithm and then suggests the best macro to be added permanently to the domain as an additional action. The generality aspects of our method are due to the use of a genetic algorithm as the learning technique and plans as the macro generation source. At one hand, genetic algorithms are automatic learning methods that can learn properties of a system using no explicit knowledge about it. Plans, at the other hand, invariably reflect successful choices of actions by the planner to cross the problem state spaces, and could bear inherently the characteristics of the planner and the domain especially that led to the solutions. Our system thus does not discover or utilise any knowledge explicitly specific to a planner or a domain.

For convenience sake, macros are represented both as sequences of constituent actions and as resultant actions built up by regression of the actions in the sequences. Macros are lifted randomly from plans of smaller problems to seed the population. Only to explore the macros occurring in plans, genetic operators are restricted to extending a macro by the preceding or the succeeding action in the plan, shrinking a macro by deletion of an action from either end, splitting a macro at one point, and lifting a macro from plans. The ranking method is based on a weighted average of the time differences while solving a different set of more difficult problems with macro augmented domains and the original domain. To show the performance of the selected individual macros, yet another set of more difficult problems are solved. We have achieved good results with two planners – FF and LPG, on five domains – gripper, tyreworld, ferry, satellite, and zenotravel. Further experiments are underway to include other planners and domains. Moreover, comparison between macros learnt by our system and those by other systems is also under consideration.

The rest of the paper is organised as follows: the next section discusses our motivations behind this work, followed by another section that describes our genetic approach of learning macros from plans; the third section onward presents our results and analyses; the last section discusses our conclusion and future works.

Motivations

Conceptually a system should achieve better performance if it can exploit its previous experiences. Our objective at the highest level is to learn experiences of a system in certain environments and to provide them somehow to the system. Moreover, we would like to achieve generality of our learning method about the systems for which it learns and about the knowledge it acquires for them.

Learning in Planning: The progress in planning research, from the earlier planners to those in the competitions, can summarily be attributed to manoeuvring diverse architectures, exploiting structural properties of the problems and translating technologies from other areas into planning. However, it has become obvious from other works carried out in parallel that planning in any realistic domain require a large volume of knowledge. Such knowledge can be acquired from properties of the planner, the domain, the problems or the plans. Planners, that incorporates learning and exploiting certain aspects, have demonstrated success but learning for planning in a generalised framework remains unexplored. Learning by the existing systems are conditional in the sense that they work only if certain properties hold for the planner or the domain. Our motivation is to develop a system that works unconditionally meaning irrespective of any particular characteristics exhibited by the planner or the domain.

Macros as Knowledge Conveyor: As noted above, macros have been used in times to encode explicit characteristics of particular domains or planners. They are, therefore, a promising means by which significant knowledge could be conveyed. Combining several steps in the state space, macros provide extended visibility of the search space to the planner. Carefully chosen macros could help find nodes that are better than the current nodes especially when the goodness of the immediate search neighbourhood cannot be measured appropriately. Macros thus could capture local search in the troublesome regions of the search space and encapsulate significant experience of the planner.

This work represents macros as normal actions and adds the best macro learnt to the domain permanently. The straightforward motivation behind this is to achieve planner independence and also to get an enhanced domain from

domain re-engineering perspective at the same time not affecting the solvability of the problem. Our method does not make any change to the planner program whereas most existing works on macros somehow need adaption or extension of the planners to support macros. However, when macros are used as additional actions, they cause more preprocessing time and incur extra overhead for the planners adding more branches in the search tree. But the latter problem is somehow minimised due to the use of a technique called helpful action pruning (Hoffmann & Nebel 2001) by most recent planners.

Macros from Plans: Plans invariably reflect successful choices of actions to cross the problem state spaces and thus could bear the characteristics of the planner and the domain inherently. For example, apparently unexplainable random action sequences in the plans could indicate confused states of the planner while it is trying to escape troublesome regions; a repeating subsequence of actions could indicate presence of structural repetitions in the domain or in the problem. Plans could, therefore, be used as a potentially useful source for macro generation. An appropriate search tool could analyse plans to produce macros that capture the choices of the planner on the problem landscapes or encode any useful domain structures. This work explores the macro space, which consists only of macros occurring in plans, to find the best individual macro that can be added to the domain permanently like a normal action.

Genetic Algorithms and Macros

The macro space seems to be restricted when macros are learnt only from plans. Yet, an exhaustive or systematic approach will not be good because macros comprising any number of actions are to be considered. This paper uses guidelines from a genetic algorithm in searching the macro space.

Genetic algorithms are automatic learning methods that require no explicit knowledge about a system, yet can discover its inherent properties. A genetic algorithm keeps a population of good individuals, generates a new population from the current one using a given set of genetic operators. It then replaces inferior current individuals by superior new individuals (if any) to get a better current population which is again used to repeat the process if the termination condition is unmet. In a particular problem context, an individual is taken for a solution (macro in our case); which means genetic algorithms are an optimisation based multi-point search on the solution space. Moreover, newly generated individuals are other possible solutions in the neighbourhood of the currently kept solutions, and a richer collection of operators explore more possible solutions. The requirements of a genetic algorithm are a suitable encoding of the individuals, a method to seed the initial population, definitions of the genetic operators to generate new individuals from the current population, and a method to evaluate individuals across the populations.

Genetic algorithms have produced promising results in learning control knowledge for domains and some success in generating and optimising plans. In (Spector 1994), Spector managed to achieve a maximally fit plan for the Sussman Anomaly, and also for a range of initial and goal states, but the problems were very small in size. SINERGY presented in (Muslea 1998) could only solve problems with specific initial and goal states. Later, GenPlan in (Westerberg & Levine 2000) showed that genetic algorithms can generate plans but when compared with the state-of-the-art planners it looks somewhat inferior. Genetic algorithms have also been used to optimise plans in (Westerberg & Levine 2001). EvoCK (Aler, Borrajo, & Isasi 2001) used heuristics generated by HAMLET (Borrajo & Veloso 1997) to seed the initial population of control rules and then genetically evolved better ones for PRODIGY4.0 (Veloso *et al.* 1995). Overall, EvoCK outperformed both PRODIGY4.0 and HAMLET. Later, L2Plan in (Levine & Humphreys 2003) used genetic approach to evolve control knowledge or policies and showed promising results by outperforming hand-coded policies.

A Genetic Algorithm on Macros

Genetic algorithms often vary in their implementations. However, the algorithm described in Figure 1 is used for this work. This section discusses its adaptation to macros.

1. Initialise the population and evaluate each individual to assign a numerical rating.
2. Repeat the following steps for a given number of epochs.
   (a) Repeat the following steps for a number equal to the population size.
      i. Generate an individual using randomly selected operators and operands, and exit if a new individual is not found in a reasonable number of attempts.
      ii. Evaluate the generated individual and assign a numerical rating.
   (b) Replace inferior current individuals by superior new individuals and exit if replacement is not satisfactory.
   (c) Exit if generation of a new individual failed.
3. Present the current best individual as the output of the algorithm.

Figure 1: A genetic algorithm

Representation of Macros

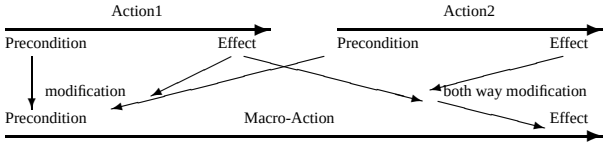
Genetic algorithms require individuals to be encoded in a composite form whereas our objective is to add macros as additional actions to the domains. Macros are, therefore, represented (see Figure 2) both as sequences of constituent actions and as resultant actions having parameters, preconditions, and effects. Given the sequence of constituent actions, the resultant action of a macro is built using *composition of actions by regression*. Genetic operators are applied on the operand macro's sequence and from the output sequence, the resultant macro's action is built.

Composition of Actions by Regression: This is a binary, non-commutative and associative operation on actions where the latter action's precondition and effect are subject to the former action's effect, and both actions' parameters are unified (see Figure 3). Every composition does not produce a valid action because the resultant precondition might have contradictions, the resultant effect might be inconsistent, and the parameters might face type conflicts while being unified. This paper considers composition of actions only in STRIPS and FLUENTS subsets of PDDL.

Precondition under Composition: A literal, appearing in the latter action's precondition, might be satisfied or contradicted and a function value might be changed by the former action's effect (see Figure 3). Therefore, the resultant

```
(:macro
    (:action move-pick-move
        :parameters (?ra ?rb - room ?b - ball ?g - gripper)
        :precondition (and
                          (at-robby ?ra)
                          (not (= ?ra ?rb))
                          (at ?b ?rb)
                          (free ?g)
                      )
        :effect       (and
                          (carry ?b ?g)
                          (not (at ?b ?rb))
                          (not (free ?g))
                      )
    )
    (:sequence
                      (move ?ra ?rb)
                      (pick ?b ?rb ?g)
                      (move ?rb ?ra)
    )
)
```

Figure 2: Representation of a macro



Param1	Param2	M.Param
?x - ball ?y - room	?y - place ?z - robby	?x - ball ?y - room ?z - robby
?x - room ?y - ball	?y - place ?z - robby	type conflict for ?y

Precond1	Effect1	Precond2	Effect2	M.Precond	M.Effect
not (p …)	(p …)	(p …)	not (p …)	true	null
(p …)	not (p …)	(p …)		false	
not (p …)	(p …)	not (p …)		false	
(p …)	not (p …)	not (p …)	(p …)	true	null
	not (p …)		not (p …)		invalid
	(p …)		(p …)		invalid
(r …)≥1	(r …) −= 1	(r …)≥2	(r …) −= 2	1≤(r …)≥3	(r …) −= 3

Figure 3: A composition of actions by regression

precondition will be a conjunct of the former action's precondition and the latter action's modified precondition.

Effect under Composition: The resultant effect will be a union of the latter action's modified effect and the former action's sub-effects which are not further modified by the latter action's effect (see Figure 3). Appropriate not-equalities are added to the precondition whenever parameters having compatible types represent different objects. This is because, when different parameters are bound with same object in similar literals or functions, it causes runtime inconsistency in the grounded effects.

Parameters under Composition: Parameters of both the actions are first unified and then union-ed together. Parameter unifications can be done by type or by name. The first option needs knowledge about the multiplicity of any static or dynamic relationships between objects; which means domain and planner characteristics are to be discovered. The second option (see Figure 3) is suitable for this work if constituent actions are lifted from plans where multiplicity issue has already been handled. In this case, problem objects are then replaced by generic variables but domain constants are left unchanged; which means actions are considered grounded partially with constants. Variables having common names are then unified replacing generalised types by specialised ones; uncommon variables however remain unaffected.

Genetic Operators

Genetic operators are restricted to extending a macro by the preceding or the succeeding action in the plan, shrinking a macro by deletion of an action from either end, and splitting a macro at a random point (see Figure 4). The motivations behind these restrictions are to explore only the macros that occur in the plans. Besides, our observations suggest subsequences of a good sequence are also good while sequences containing bad subsequences are also bad. Moreover, lifting of random sequences from plans is also used as yet another operator to facilitate diversity of the macro space exploration.

Plan	…(a …)(b …)(c …)(d …)(e …)(f …)(g …) …(j …)(k …)(l …)(m …) …			
Macro	(b …)(c …)(d …)(e …)(f …)			
Extend	(a …)(b …)(c …)(d …)(e …)(f …)		(b …)(c …)(d …)(e …)(f …)(g …)	
Shrink	(c …)(d …)(e …)(f …)		(b …)(c …)(d …)(e …)	
Split	(b …)(c …)(d …)	(e …)(f …)	(b …)(c …)	(d …)(e …)(f …)
Lift	(j …)(k …)(l …)			

Figure 4: Genetic operators

Seeding the Initial Population

To seed the initial population, sequences of actions are randomly lifted (using the *lift* operator) from plans of smaller problems called *seeding problems*.

Evaluating Individual Macros

To evaluate individual macros, a different set of problems called *ranking problems* are used. These problems are more complex than the seeding problems. However, they do not take much planning time since they are solved for every macro. For each macro, an *augmented domain* is produced adding it as an additional action to the original domain. The ranking problems are then solved both with the original domain and the augmented domain and the utility function in Figure 5 is used to assign the macro a numerical rating.

Utility Function: The basic form of the utility function is U which is a weighted mean time gain. The multiplicative factors, *i.e.* probabilities, are to counterbalance the effect of any misleadingly high value of U. For a good macro, most problems (reflected by P_s) should be solved within given limits taking less time in most cases (reflected by P_g). Macro usage statistic is not considered because there could be less frequent tricky macros that save enormous search time.

1. Deterministic Planners: The gain for a problem is the difference of the time taken to solve it using the original domain and that using the augmented domain. The weight of a problem is the time taken to solve it using the original domain after being normalised by the total time taken to solve all the problems using the original domain.
2. Stochastic Planners: The time taken to solve a problem by such a planner is randomly distributed. Observations suggest the underlying distributions to be normal. A problem is therefore solved a number of times and a random variable having parameters (mean μ, variance σ^2, sample count n) is used to represent the time distribution. The t-value by Student's t test, which computes the significance of difference between two normal distributions, is then used as a difference operator to measure the gain. The statistical sum of the time distributions for all the problems using the original domain is used in normalisation. The time distributions, the individuals and the sum, are then

replaced by their respective mean-to-error ratio to come up with a weight for a problem. Use of mean-to-error ratio has another implication that narrowly dispersed distributions get more weight.

$$
\begin{aligned}
Utility &= UP_sP_g && \text{if } U > 0 \text{ and } P_sP_g > 0\\
&= \tfrac{U}{P_sP_g} && \text{if } U < 0 \text{ and } P_sP_g > 0\\
&= -e^{\alpha} && \text{if } P_s = 0\ (e^{\alpha} \to \infty)\\
&= UP_se^{-\alpha/2} && \text{if } U > 0 \text{ and } P_g = 0\\
&= \tfrac{U}{P_s}e^{\alpha/2} && \text{if } U < 0 \text{ and } P_g = 0\\
U &= \textstyle\sum_{p\in S} G_pW_p && S\text{: set of problems}\\
G_p &= t_{op} - t_{ap} && \text{if deterministic}\\
&= t(T_{op}, T_{ap}) && \text{if stochastic}\\
W_p &= t_{op}/\textstyle\sum_{q\in S} t_{oq} && \text{if deterministic}\\
&= w(T_{op}, \textstyle\sum_{q\in S} T_{oq}) && \text{if stochastic}\\
T_s &= \textstyle\sum_{q\in S} T_{oq} && \text{(sum of distributions)}\\
&= (\textstyle\sum_{q\in S}\mu_{oq}, \sum_{q\in S}\sigma^2_{oq}, \sum_{q\in S} n_{oq})\\
t(T_o, T_a) &= (\mu_o - \mu_a)/\sqrt{\tfrac{\sigma^2_o}{n_o} + \tfrac{\sigma^2_a}{n_a}} && \text{(Student's t test)}\\
w(T_q, T_s) &= \tfrac{\mu_q}{\sigma_q/\sqrt{n_q}} / \tfrac{\mu_s}{\sigma_s/\sqrt{n_s}} && \text{(using mean-to-error ratio)}
\end{aligned}
$$

where

P_s: Probability that a problem is solved using the macro within given limits.

P_g: Probability that time-gain is achieved when a problem is solved using the macro. For stochastic planners, t test with $\alpha = 0.05$ determines time-gain.

$t_{op}, T_{op}(\mu_{op}, \sigma^2_{op}, n_{op})$: time to solve Problem p using no macro.

$t_{ap}, T_{ap}(\mu_{ap}, \sigma^2_{ap}, n_{ap})$: time to solve Problem p using the macro.

Figure 5: A utility function for macro evaluation

Pruning the Macro Space

We adopt some pruning techniques to reduce any effort wasted otherwise to explore potentially inferior macros.

Pruning during generation: The following strategies help prune macros in Step 2(a)i of the algorithm in Figure 1:

1. Macros having unsatisfiable preconditions or inconsistent effects are discarded whenever detected.
2. Actions having no parameter in common (by name) generally make no sensible macro. This also prevents irrelevant actions are not part of a macro.
3. The more the parameters, the more the unnecessary instantiated operators. This at least affects the planners which do operator instantiation in their preprocessing steps.
4. Longer sequences of actions are more specific to certain objectives and are less likely to be useful for wider ranges of problems.
5. Sequences of actions that have subsequences producing null effects, are not minimal.
6. Similar sequences of actions differing only by parameterisation are considered copies of a single sequence.
7. Sequence of actions equivalent in partial order are considered copies of a single sequence. This technique however is under implementation and has not been used for this paper.
8. A strategy that consecutive actions in a macro must have a causal link between them is considered but finally has not been used. This is because there could be an auto correlation between actions such that execution of them together somehow helps the planner solve problems faster. The auto correlation could be inherent in the planner's architecture or implementation, or could be in the domain model as well.

Pruning during evaluation: Early detection of inferior macros in Step 2(a)ii in Figure 1 saves learning time needed otherwise to solve the remaining problems.

1. Failure to solve a problem using the augmented domain within certain limits whereas it is solvable using the original domain implies the macro is causing much overhead and resource (time, memory, etc.) scarcity to the planner.
2. Macros that are not used in any plans are not of any interest. They could have unsatisfiable preconditions which could not be detected in the generation phase or they are simply not helpful to the planner.
3. Macros that cause invalid plans to be generated have inconsistencies in their effects. Inconsistent effects that arise during the runtime of a planner, most probably due to the way it handles parameter binding, cannot completely be detected in the generation phase.

Experiments

To analyse performances of the macros suggested by our system, a third different set of problems called *testing problems* are used. These problems are more difficult than the ranking problems in terms of the problem parameters and the time required to solve them using the original domain. For a suggested macro, the testing problems are solved using both the original domain and the augmented domain.

Results

Figure 6 describes the typical setup of our experiment. Figure 7 then shows the performances of the suggested macros for the planners FF and LPG on the domains Ferry, Gripper, Tyreworld, Zenotravel, and Satellite. Moreover, Figure 8– 16 show the plan times graphically. On the charts, *domain* and *domain-*k respectively denote the original domain and the augmented domain for the kth best macro ($0 \leq k \leq 2, k = 0$ for the best). To clarify certain things, note that the utility values of the third macros in Figure 8 and 11 are very low. They are included to show that some macros could be good for smaller problems but performance could degrade as problems get larger. One major pitfall observed in Figure 14 is that the best macro, as suggested by the our method, unexpectedly does not perform well.

Analyses

For comprehensive analyses of this work, our qualitative achievements are presented as hypotheses which are then justified by our results.

Hypothesis 1 *Our utility function is consistent across given problems, domains and planners.*

* Planners: FF, LPG
* Domains: Gripper, Ferry, Tyreworld, Zenotravel, Satellite
* Number of random problems: Seeding 3–5, Ranking 6–20, Testing 12–20
* Macro size limits: Maximum parameters 8, Maximum sequence length 8
* Operator selection probability: Extend 30%, Shrink 30%, Split 30%, Lift 10%
* Sample count for a stochastic planner to represent the distribution: 10
* Evaluation phase pruning: a macro is pruned out if more than 40% problems or runs are unsatisfactory
* Current and child population: 1–2 × number of actions
* Maximum number of epochs to run: 15-30
* Replacement: at least 1 in every 3 consecutive epochs
* Generation attempts: maximum 20000 for every new macro

Figure 6: Experimental setup

Macro	Plan Time Gain %		Plan Length Loss %		Macro Occurrence %	
	gain[1]	prob[3]	loss[2]	prob[3]	usage[2]	prob[3]
FF-Ferry-0	79	100	31	100	33	100
FF-Ferry-1	79	100	31	100	33	100
FF-Ferry-2	27	70	31	100	33	100
LPG-Ferry-0	78	100	09	100	168	100
LPG-Ferry-1	79	100	09	100	67	100
LPG-Ferry-2	40	90	09	100	116	100
FF-Gripper-0	89	100	33	100	33	100
FF-Gripper-1	72	100	33	100	33	100
FF-Gripper-2	76	100	33	100	33	100
LPG-Gripper-0	94	100	26	100	474	100
LPG-Gripper-1	55	100	22	100	86	100
LPG-Gripper-2	07	100	11	100	28	100
FF-Tyreworld-0	83	100	00	00	09	100
FF-Tyreworld-1	78	100	00	00	09	100
FF-Tyreworld-2	77	100	00	00	09	100
LPG-Tyreworld-0	30	100	-01	58	45	100
LPG-Tyreworld-1	21	87	10	100	76	100
LPG-Tyreworld-2	09	54	00	90	44	100
FF-Zenotravel-0	-96	15	34	100	31	100
FF-Zenotravel-1	49	95	03	87	10	100
LPG-Zenotravel-0	26	90	10	100	41	100
LPG-Zenotravel-1	33	97	14	100	72	100
LPG-Zenotravel-2	29	97	-01	90	73	100
FF-Satellite-0	79	100	27	80	53	100
FF-Satellite-1	71	100	73	100	53	100
LPG-Satellite	no macro with positive utility could be learned					

1 with respect to the plan time using the original domain
2 with respect to the plan length using the original domain
3 prob stands for probability

Figure 7: Performance: time gain, quality loss, usage

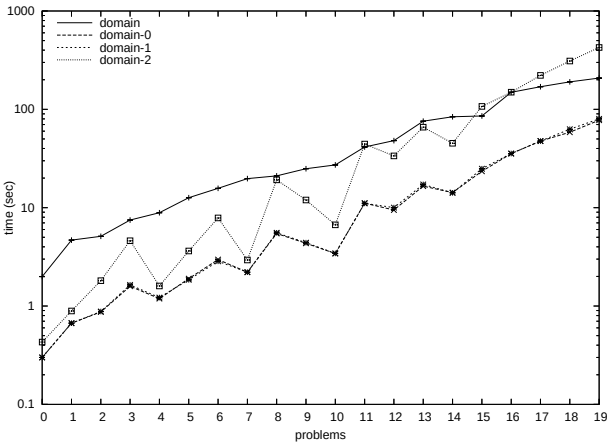


Figure 8: Plan time: FF-Ferry

Justification: For most macros in most domains, the utility values computed against the ranking problems and the testing problems (which are more difficult) are found correlated. One major violation found is with the best suggested macro in the domain zenotravel for the planner FF. The third suggested macros in Figure 8, 10, and 11, also show minor violations; but note that their utility values are very small. Due to the variation in computation of gain and weight for deterministic and stochastic planners and due to the use of only one planner for each category, the claim against the planners could appear unjustified. But note that the basic notion behind all such computation is same. Moreover, experiments with further planners are underway. ■

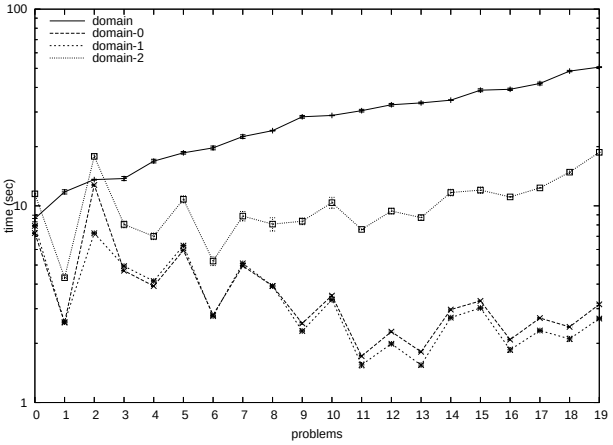


Figure 9: Plan time: LPG-Ferry

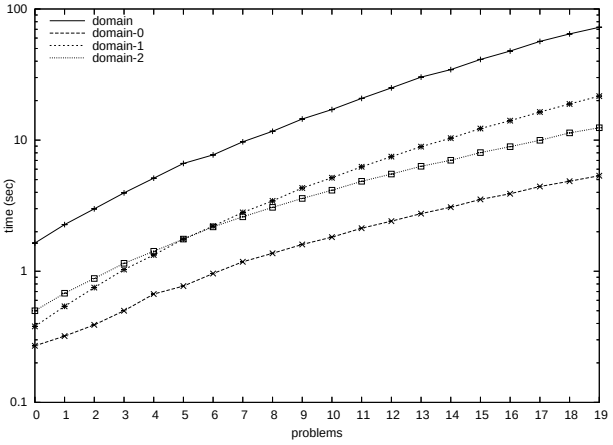


Figure 10: Plan time: FF-Griper

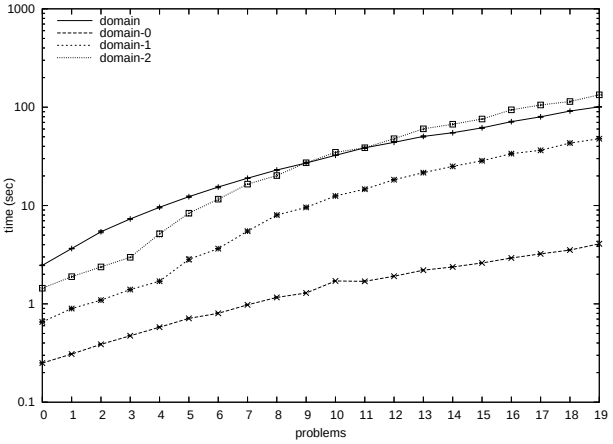


Figure 11: Plan time: LPG-Griper

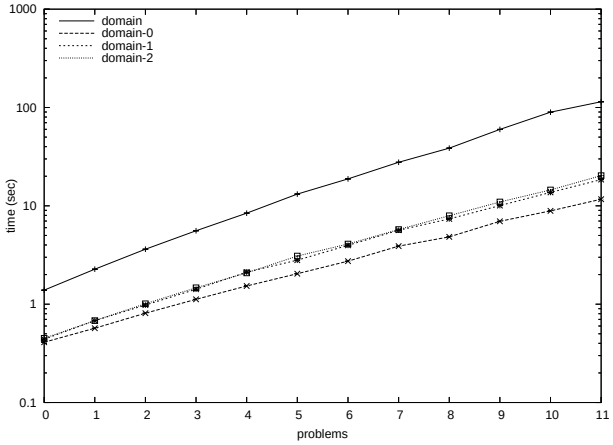


Figure 12: Plan time: FF-Tyreworld

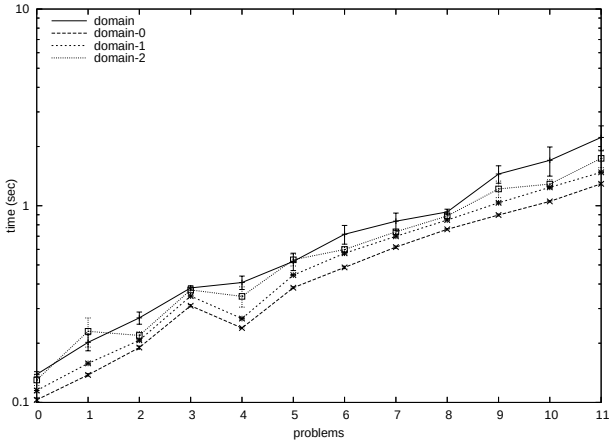


Figure 13: Plan time: LPG-Tyreworld

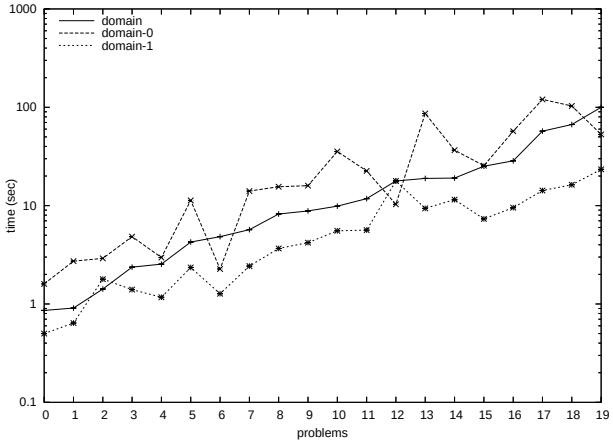


Figure 14: Plan time: FF-Zenotravel

Hypothesis 2 *Macros can be learnt effectively from plans without knowing anything explicitly specific about planners, domains, problems, or plans.*

Justification: Our method does not strive to discover or utilise any explicitly specific properties from either of planners, domains, problems or plans. Moreover, It only explores only the macros that occurs in plans. We so far have managed to learn macros effectively for two planners on most of the five domains. ■

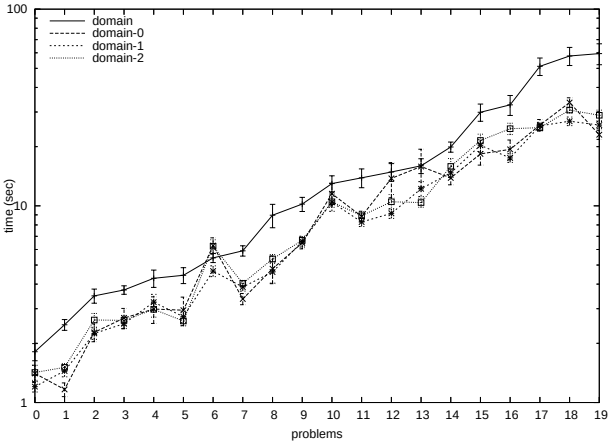


Figure 15: Plan time: LPG-Zenotravel

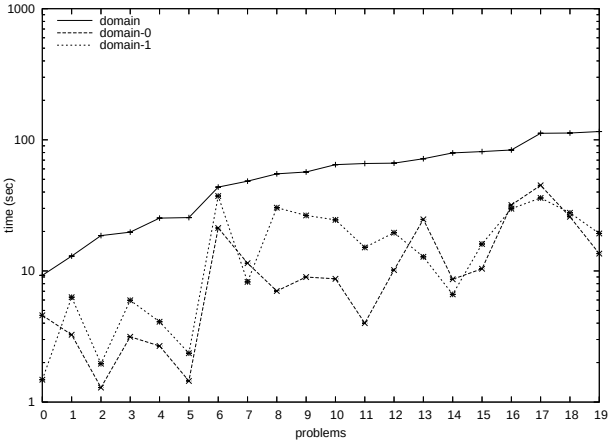


Figure 16: Plan time: FF-Satellite

Other observations that appear straightforward from our experiments are as follows:

1. The best macros learnt from a domain for different planners are not always same, but sets of top performing macros seem overlapping. More experiments are needed to go for a conclusion.
2. Macros usually lead to longer plans as observed with most domains and both planners.
3. Our work generalises macro learning from plans since it does not look for any particular properties.

At the end, learning by our method depends on the problems it is provided with. But this is a common issue with any such or similar approach. Currently, we use randomly generated problems, but in future we would like to work on what problems could be used to learn macros effectively.

Conclusion

This paper presents an automatic macro learning method that requires no structural knowledge about the domains and the planners. Despite recent significant progress in planning, many complex domains, and even simple domains with larger problems, remain challenging. Macros provide

a promising avenue in planning research to achieve further improvement. Macros when added to the domain like normal actions can convey significant knowledge to the planner. Experiences of the planner on the problem landscape can thus be encoded while re-engineering a domain. Most existing work on macros somehow need knowledge specific to the planner and the domain. A macro learning method that does not need such knowledge, therefore, gets importance. Our method learns macros effectively from plans using a genetic algorithm. Genetic algorithms are automatic learning methods that can discover inherent characteristics of a system using no explicit knowledge about it. We have achieved convincing, in some cases dramatic, improvement with two planners – FF and LPG, on five domains – gripper, tyreworld, ferry, satellite, and zenotravel. Further experiments to learn macros for different planning systems and for more complex and numerical domains are underway. As we consider only individual macros for the time being, we hope to extend our approach to learn a set of macros either incrementally or using a genetic approach on macro-sets. It would also be useful to compare performances of our system and other related systems.

Acknowledgement

This research is supported by the Commonwealth Scholarship Commission in the United Kingdom.

References

Aler, R.; Borrajo, D.; and Isasi, P. 2001. Learning to solve problems efficiently by means of genetic programming. *Evolutionary Computation* 9(4):387–420.

Armano, G.; Cherchi, G.; and Vargiu, E. 2005. A system for generating macro-operators from static domain analysis. In *Proceeding (453) Artificial Intelligence and Applications*.

Bacchus, F. 2001. AIPS'00 planning competition. *AI Magazine* 22(3):47–56.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review* 11(1–5):371–405.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Coles, A., and Smith, A. 2004. MARVIN: Macro-actions from reduced versions of the instance. In *IPC4 Booklet*. ICAPS.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 465–471.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Gerevini, A., and Serina, I. 2002. LPG: a planner based on local search for planning graphs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*.

Hernádvölgyi, I. 2001. Searching for macro operators with automatically generated heuristics. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, 194–203.

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.; Edelkamp, S.; Englert, R.; Liporace, F.; Thiébaux, S.; and Trüg, S. 2004. Towards realistic benchmarks for planning: the domains used in the classical part of IPC-04. In *4th International Planning Competition*, 7–14. ICAPS.

Iba, G. A. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3:285–317.

Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26:35–77.

Levine, J., and Humphreys, D. 2003. Learning action strategies for planning domains using genetic programming. In *Applications of Evolutionary Computing, EvoWorkshops2003*, volume 2611, 684–695.

Long, D., and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research* 20(Special Issue on the 3rd International Planning Competition):1–59.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2):35–55.

Minton, S. 1985. Selectively generalising plans for problem-solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Muslea, I. 1998. A general purpose AI planning system based on the genetic programming paradigm. In *Proceedings of the World Automation Congress*.

Newell, A., and Simon, H. 1972. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newton, M. A. H.; Levine, J.; and Fox, M. 2005. Genetically evolved macro-actioins in ai planning. In *24th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*.

Spector, L. 1994. Genetic programming and AI planning system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, 1329–1334.

Veloso, M.; Carbonell, J.; Perez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7:81–120.

Westerberg, C. H., and Levine, J. 2000. GenPlan: Combining genetic programming and planning. In *19th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*.

Westerberg, C. H., and Levine, J. 2001. Optimising plans using genetic programming. In *6th European Conference on Planning*.