Evolving Macro-Actions for Planning

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

Abstract

Domain re-engineering through macro-actions (*i.e.* macros) provides one potential avenue for research into learning for planning. However, most existing work learns macros that are reusable plan fragments and so observable from planner behaviours online or plan characteristics offline. Also, there are learning methods that learn macros from domain analysis. Nevertheless, most of these methods explore restricted macro spaces and exploit specific features of planners or domains. But, the learning examples, especially that are used to acquire previous experiences, might not cover many aspects of the system, or might not always reflect that better choices have been made during the search. Moreover, any specific properties are not likely to be common with many planners or domains. This paper presents an offline evolutionary method that learns macros for arbitrary planners and domains. Our method explores a wider macro space and learns macros that are somehow not observable from the examples. Our method also represents a generalised macro learning framework as it does not discover or utilise any specific structural properties of planners or domains.

Introduction

Planning has achieved significant progress in recent years from planning competitions. The focus of planning research, however, lies mostly on developing planning technologies while the impact of problem formulation on its solution process remains overlooked. Re-engineering a domain by utilising knowledge acquired for a planner paves the way for further research in this direction. Macro-actions, when represented as additional actions, are one relatively convenient way by which to convey such knowledge and achieve domain enhancements. Within current limits of the Planning Domain Definition Language (PDDL), any knowledge can be conveyed only by additional actions and practically only in STRIPS and FLUENTS subsets of the PDDL.

A *macro-action*, or *macro*, is a group of actions selected for application at one time like a single action. So, one application of a macro leads to planning of several steps at a time. Macros could represent plan fragments that are found with enormous search effort or are frequently used. Macros could capture local search to find better successor nodes especially when the immediate search neighbourhood is not good. Macros could affect neighbourhood evaluation and thus take the search in a different direction. Consequently, a goal could be *reached* quickly and problems that are *unsolvable*[1] could become *solvable*. When macros are added into a domain as additional actions, no planner modification is needed; also, the *reachability* of a problem is not affected. But they cause more preprocessing time and incur an extra overhead for the planners adding more branches in the search tree. However, the latter problem is minimised due to the use of a technique called *helpful action pruning* (Hoffmann & Nebel 2001) by many recent planners.

Motivations. Most existing work learns macros from domain analysis or reusable plan fragments (see the section describing related work). Only macros that encapsulate static domain properties can be learnt from domain analysis. The reusable plan fragments can be captured from online planner behaviours or offline plan characteristics. Although our other work in (Newton *et al.* 2007) showed various macros that improve performance could be learnt successfully from plans using no specific structural knowledge, most existing work somehow exploits particular domain or planner properties. However, during search, the planners often get overwhelmed by the numerous grounded actions available at each node. In these situations, they adopt pruning strategies that help them select next steps very quickly. While making such cursory moves, the chance of not considering all potentially good options remains high. Such hasty decisions thus do not always reflect the better choices available. Furthermore, the example problems, used for macro acquisition, might not cover many aspects of the system as what problems make a good collection is not easily addressed. Nevertheless, most of the existing methods explore macro spaces that are, in effect, restricted. Therefore, the main motivation for this work is to explore the macros that are normally not considered and so not observable. An evolutionary algorithm with its rich operator collection and search direction could help much in this regard. Another motivation of this work is to maintain the generality about planners and domains that our aforementioned work obtained. This is because any specific properties are not likely to be common with a wider range of planners or domains.

[1] By *solvability* we mean, using the original domain, whether the planner can solve the problem within given resource (*e.g.* time, memory, etc.) limits. Whether the goal of a problem can be attained in a given context is discussed under the term *reachability*.

Contribution. This paper presents an offline evolutionary macro learning method that works with arbitrary planners and domains. Given a planner, a domain, and a number of example problems, our method learns and suggests individual macros that are to be included permanently in the domain as additional actions. The main highlight however is this method can learn macros that are not observable from examples; the reasons could be they are not normally considered and/or the examples do not cover the aspects. This work successfully shows that such macros are also useful and in many cases, are suggested, based on our evaluation, over the observable macros. Another highlight is this method, unlike existing work, does not exploit any specific structural properties of planners or domains. The desirable aspects of our method are due to the use of an evolutionary approach as our learning technique and plans as the source of constituent actions. On one hand, plans invariably reflect successful choices of actions by the planner and so bear inherently the characteristics of the planner or the domain especially that led to the solution. Evolutionary algorithms, on the other hand, are automatic learning methods that not only can capture observable features of a system using no explicitly specific knowledge about it, but also can evolve other inherent features. We have achieved convincing results with several planners and domains.

Overview. For the sake of convenience, macros are represented both as sequences of parameterised (and so generalised) constituent actions and as resultant actions built up by regression of the actions in the sequences. While building macros from scratch, or lifting from plans, or evolving from other macros, plans of *smaller problems*[2], called *seeding problems*, are used as the source of constituent actions. A rich collection of genetic operators is used to explore a wider macro space which includes observable macros as well. The evaluation of macros is done against other *larger* but solvable problems, called *ranking problems*. These problems are, however, not too large because they are attempted to be solved for every macro. Also, they are not too small because time gains cannot be measured properly for smaller problems. The fitness function is based on a weighted average of the time gains while solving the ranking problems with the macro augmented domain and the original domain. After the learning is accomplished, yet another set of more difficult problems (which might include unsolvable instances), called *testing problems*, are used to demonstrate the performance of the selected individual macros.

The rest of the paper is organised as follows: the next three sections discuss evolutionary algorithms, related work, and an evolutionary approach of learning macros, the last two sections discuss our experiments and conclusion.

Evolutionary Algorithms

An evolutionary 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 until the termination condition is met. In a particular problem context, an individual is taken for a solution (macro in our case); which means evolutionary 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 genetic operators explore more possible solutions. The requirements of an evolutionary 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. Note that, by satisfying such requirements, the specific knowledge, we give, is actually generic in planning and by no way specific to a planner or a domain.

Evolutionary algorithms have produced promising results in learning control knowledge for domains and some success in generating plans. EvoCK (Aler, Borrajo, & Isasi 2001) evolved heuristics generated by HAMLET (Borrajo & Veloso 1997) for PRODIGY4.0 (Veloso *et al.* 1995) and outperformed both of them. L2Plan (Levine & Humphreys 2003) evolved control knowledge or policies that outperformed hand-coded policies. Spector, using evolutionary algorithms, managed to achieve plans for small problems having a range of initial and goal states (Spector 1994). SINERGY (Muslea 1998) could only solve problems with specific initial and goal states. GenPlan (Westerberg & Levine 2000) showed that evolutionary algorithms can generate plans; but it is somewhat inferior than the state-of-the-art planners. Recently, we have used an evolutionary approach successfully to learn macros from plans for arbitrary plans and domains (Newton *et al.* 2007). Evolutionary algorithms have also been used to optimise plans in (Westerberg & Levine 2001).

Related Work

Macros are not very new in planning research. STRIPS (Fikes, Hart, & Nilsson 1972) generates macros from unique subsequences of wholly parameterised plans. REFLECT's (Dawson & Siklóssy 1977) macros are based on causal links between actions in the domain. MORRIS (Minton 1985) learns macros from plan fragments that are frequently used or achieve interacting goals. Macro Problem Solver (MPS) (Korf 1985) learns a complete set of macros that totally eliminates the search but only for a particular goal in fixed size problems of domains that exhibit operator decomposability. MACLEARN (Iba 1989) learns macros from action sequences that lead the search to reach a peak from another peak in its heuristic profile. It then uses an automated static filter based on domain knowledge and a manual dynamic filter based on usages of macros in plans. MARVIN (Coles & Smith 2007) learns macros from the plan of a reduced version of the given problem after eliminating symmetries and also from the action sequences that help the search escape plateaus in its heuristic profile. Macro-FF (Botea *et al.* 2005) learns macros by using component level abstraction based on static facts of a domain and also by partial-order lifting from plans based on an analysis of causal links. It then evaluates the macros by solving other problems and counting the number of states explored. A very preliminary

[2]By *problem size or difficulty level* we mean, the time required by the given planner to solve the problem with the original domain.

stage of this work is reported in (Newton, Levine, & Fox 2005). Also, a similar approach (Newton *et al.* 2007) is presented later that learns macros occurring in plans only for arbitrary planners and domains. However, this work is different from any of the above in exploring the space of non-observable macros and showing that such macros could be suggested over observable ones.

An Evolutionary Macro Learning Method

Our learning method is described in Figure 1 where individuals are taken as macros. Its implementation issues include representation, generation, and evaluation of macros along with validation and pruning techniques to reduce any effort wastage. Because of space constraints, we describe them very briefly. For further details, we refer the reader to our other work reported in (Newton *et al.* 2007). These two implementations only differ in genetic operators used and pruning strategies adopted. This work has a rich collection of operators and uses pruning strategies as few as possible to facilitate exploration of a wider macro space. In contrast, the other work has a restricted operator set and more stringent pruning rules to explore macros occurring in plans only.

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. Suggest the best individuals as the output of the algorithm.

Figure 1: An evolutionary learning method

Macro Representation. Macros are represented both as sequences of parameterised (and so generalised) constituent actions and as resultant actions composed up by *regression*[3] of actions in the sequences. Genetic operators are applied on the operand macro's sequence and from the output sequence, the resultant macro's action is built. Had PDDL supported macros, the action composition would not be required and planners could easily execute a macro sequence.

Macro Generation. Macros are generated by using the genetic operators described in Figure 2. The operand actions come from plans of the seeding problems and the macros from the current population. All operands are, however, selected randomly. To seed the initial macro population, only *lift* and *construct* operators are used. Note, a subset of these operators, that includes lift, extend, delete only at ends, and split, explores macros observable from plans and thus occasionally drives towards convergence.

Macro Evaluation. For each macro, an *augmented domain* is produced adding it as an additional action to the original domain. For all the ranking problems, the planner is then run both with the original domain and the augmented domain under similar resource limits. The utility function, shown in Figure 3 is then used to give a numerical rating to the augmented domain (and so the macro) against the original domain. For a good macro, in qualitative terms, *most problems* (reflect by C) should be solved taking *less time* (reflected by S) in *most cases* (reflected by P) with its augmented domain. A bad macro, in contrast, would cause an overhead that leads to longer solution times or even failures in solving problems within given resource limits. Good macros, however, may not have *high usage* because less frequent but *tricky* macros could save enormous search time. The utility function is mainly based on S which is a weighted average of time gains giving larger problems more weight. The other factors are to counterbalance any misleadingly high value.

- ⋆ *Lift*: an action sequence is randomly lifted as it appears in a plan.
- ⋆ *Construct*: an action sequence is built from scratch by picking individual actions up randomly.
- • *Extend*: an action appearing immediate next/prior to a macro is appended at the respective end.
- • *Insert*: an action is inserted at a random position (including ends) of a macro.
- • *Delete*: an action is deleted from a random position (including ends) of a macro.
- • *Alter*: an action replaces one random action of a macro.
- • *Split*: a macro is split at one random position and either one gives the output macro.
- ◦ *Merge*: two macro sequences are concatenated together to produce one output macro.
- ◦ *Crossover*: a random prefix of one macro is concatenated with a random suffix of another macro.

Figure 2: Description of the genetic operators

Macro Pruning. A number of strategies are adopted in Step 2(a)i of the learning method in Figure 1; these prune generated macros before evaluation. Action sequences that have subsequences producing null effects are not minimal. Action sequences, differing only by parameterisation or equivalent in partial order, are considered as same sequence. Maximum sequence length and maximum parameter count are fixed by given bounds. Early detection of inferior macros during evaluation in Step 2(a)ii in Figure 1 saves learning time needed otherwise to solve the remaining problems. 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.

Other stringent strategies like *causal links* or *common parameters* between constituent actions are not adopted although they ensure cohesiveness of constituent actions and also oversee irrelevant actions are not part of a macro. This is because our interest is to learn any macros that help speedup the search. A macro might have some helpful auto correlations between its constituent actions. Even a macro not usable in any plans might be helpful in the search.

Macro Validation. Pruning strategies help detect invalid macros during their generation and also during their evaluation. Furthermore, plans produced with the augmented domains are validated as needed to detect macros that somehow cause invalid plans to be produced by the planner.

[3] Action composition by regression is a binary, associative, and non-commutative 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. Regression is practically feasible in STRIPS and FLUENTS only.

$U = \begin{cases} C \times S \times P \\ = -\frac{1}{2} \text{ if } C = 0 \\ = -1 \text{ if invalid plans produced} \end{cases}$

$C = \Sigma_{k=1}^{n} c_k / n$

$S = w\Sigma_{k=1}^{n} s_k w_k + w' \Sigma_{k=1}^{n} s'_k w'_k$

$P = \Sigma_{k=1}^{n} p_k$

Where,

n: Number of ranking problems to be solved.

m: Number of times a ranking problem is to be solved. For a deterministic planner, $m = 1$. A stochastic planner produces different plans in different runs taking different times. Therefore, larger m, preferably ≥ 5, gives a time distribution t(sample-count ν, mean μ, dispersion $\delta = \sigma/\sqrt{\nu}$) for such planners.

$t_k(\nu_k, \mu_k, \delta_k)$: Time distribution for problem-k while solving with the original domain. Note, each problem is solved m times with the original domain *i.e.*, $\nu_k = m$. Moreover, $\mu_k > 0$. When $m = 1$, $\nu_k = 1$ and so $\delta_k = 0$. If $\delta_k = 0$, any terms involving δ_k are omitted.

$t'_k(\nu'_k, \mu'_k, \delta'_k)$: Time distribution for problem-k while solving with the augmented domain. Note, $0 \leq \nu'_k \leq m$. When the problem is not solved (*i.e.*, $\nu'_k = 0$), $\mu'_k = \infty$. When $m = 1$, $\nu'_k = 0$ or 1 and so $\delta'_k = 0$.

$t(\nu, \mu, \delta) = \Sigma_{k=1}^{n} t_k$: Total time distribution for all the ranking problems while solving with the original domain. This is a sum of random variables. Therefore, $\nu = \Sigma_{k=1}^{n} \nu_k = mn$, $\mu = \Sigma_{k=1}^{n} \mu_k$, and $\delta^2 = \Sigma_{k=1}^{n} \delta_k^2$.

$c_k = \nu'_k / \nu_k$: Probability that problem-k is solved using the augmented domain.

$s_k = \mu_k / (\mu_k + \mu'_k)$: The normalised gain/loss in mean while solving problem-k with the augmented domain. Note, $s_k = 1$, $\frac{1}{2}$, and 0 for $\mu'_k = 0$, μ_k, and ∞.

$s'_k = \delta_k / (\delta_k + \delta'_k)$: The normalised gain/loss in dispersion while solving problem-k with the augmented domain. If $m = 1$, s'_k is defined to be 0 and omitted as $\delta_k = \delta'_k = 0$. Note, $s'_k = 1$, $\frac{1}{2}$, and 0 for $\delta'_k = 0$, δ_k, and ∞.

$w_k = \mu_k / \mu$: Weight of gain/loss in mean giving more emphasis on larger problems

$w'_k = 1/n$: Weight of gain/loss in dispersion giving equal emphasis on all problems

$w = \mu/(\mu + \delta)$: The overall weight of gain/loss in mean.

$w' = \delta/(\mu + \delta)$: The overall weight of gain/loss in dispersion.

$p_k =$ 1 for gain, 0 for loss, $\frac{1}{2}$ otherwise. The Students t-test at 5% significance level on t_k and t'_k determines a gain or a loss. Alternatively, $\text{sign}(\mu_k - \mu'_k)$ is used when $m = 1$ and/or t-test cannot be used because δs are zero.

Figure 3: A utility function for macro evaluation

Experiments

To demonstrate success for arbitrary planners, we choose several planners – FF, VHPOP, SGPLAN, and Fast Downward (FD). The planners have different basic characteristics and are current state-of-the-art ones in their respective tracks. The domains chosen are some new domains written based on bench mark domains used in planning research (see Figure 4) although we have some results directly on existing domains like Blocks, Satellite, and Ferry. This is because planners could be over-fitted with the benchmark domains and we wanted to investigate how they perform on newer domains with or without macros. Another reason is most candidate bench mark domains are smaller and simpler (*e.g.* transportation domains) or too large (*e.g.* Settlers). Nevertheless, the new domains pose sufficient challenge to the state-of-the-art planners as seen from Figure 6 and 7. As noted before, the problems used to demonstrate performance of the suggested macros are the testing problems. For a suggested macro, the testing problems are solved using both the original domain and the augmented domain.

Results. Figure 5 describes the typical setup of our experiments. The parameter values in most cases are chosen intuitively. Because of space constraints, Figure 6 shows the plan times of some of the macros graphically for quick understanding. However, Figure 7 summarises performances of the suggested macros for all planner-domain pairs. Note,

- *Blocks* domain has an arm that picks and drops blocks to build stacks on a table.
- *Satellite* domain deals with a number of satellites that can take images of targets in various modes and transmit data to the base. PSatellite and NSatellite are its propositional and numerical versions. The numerical version has additional constraints on buffer capacity.
- *Railway* domain (10 actions) is a reduced version of the Settlers domain. Only the railway construction part has been taken with relevant structures added. PRailway and NRailway represents its propositional and numerical models respectively.
- *Reliefwork* (12 actions) is a transportation domain based on disaster management scenarios in flood affected areas. Victims are to be attended by a patrol boat. Depending on a victim's need, a relief pack is delivered, or a pickup boat is called to take him/her to a shelter, or an ambulance boat is called to take the victim to a hospital. PReliefwork and NReliefwork are respectively the propositional and numerical versions.
- *NCokeCake* domain (10 actions) combines two problems – pouring and filling jugs of certain volumes to measure any volume and putting fixed weights on and off on a scale both additively and subtractively to measure any weight.

Figure 4: Domains used in this work

all the macros presented here (in first five charts) are non-observable and are better (as our learning method suggests) than the best observable macros. For the time being, we put emphasis on the prospect of exploring non-observable macros and do not show their performance comparison against observable ones; however, the last chart in Figure 6 shows such results in PSatellite domain for planner VHPOP.

- ⋆ Number of random problems: Seeding 5, Ranking 20, Testing 50
- ⋆ Macro size limits: Maximum parameters 12, Maximum sequence length 16
- ⋆ Operator selection probability: around 10% for each operator
- ⋆ Sample count for a stochastic planner to represent the distribution: 5
- ⋆ Evaluation phase pruning: a macro is pruned out if more than 50% problems or runs are unsatisfactory
- ⋆ Number of epochs: 200 Population size: 2 × number of actions
- ⋆ Satisfactory replacement level: at least 1 in every 50 consecutive epochs
- ⋆ Generation attempts: maximum 999999 for every new macro
- ⋆ Limits: memory 1 gigabyte, time (secs)- seeding 10, ranking 20, testing 1800
- ⋆ Computers' configuration: Pentium 4, Linux, CPU 3ghz, RAM 2gb

Figure 5: Experimental setup

Analysis. For a comprehensive analysis of this work, the qualitative achievements are presented as hypotheses with proper justification made by the results.

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

Justification: As mentioned earlier, the exact utility values assigned to the macros are relative to the ranking problems and the planner used. Therefore, it is necessary to show the qualitative consistency of our utility function. For this, we computed utility values of the suggested macros against the testing problems. For most macros in most domains, these values are positively correlated with the values assigned against ranking problems during evaluation. Furthermore, many of the suggested macros (see Figure 7) achieve significant improvement with the testing problems. ■

Hypothesis 2 *Without exploiting any specific knowledge about planners or domains, our method can effectively learn macros that are not observable from plans.*

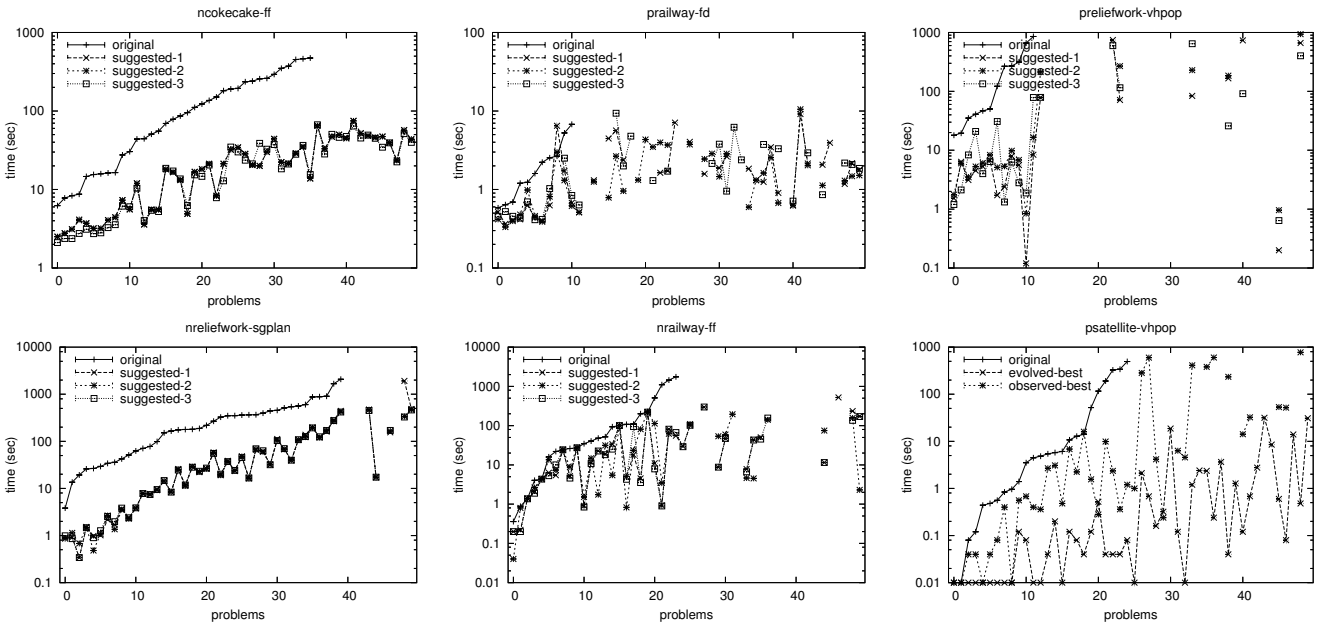

Figure 6: Time performance of some suggested macros

- S% problems are solved only with the augmented domain and s% only with the original domain.
- T% problems take less time with the augmented domain and t% with the original domain.
- L% problems have less plan length with the augmented domain and l% with the original domain.
- (P%, p%) is (mean, dispersion) of plan time (T) performance $(T_{\mathrm{Orig}} - T_{\mathrm{Aug}})/T_{\mathrm{Orig}}$
- (Q%, q%) is (mean, dispersion) of plan length (L) quality $(L_{\mathrm{Orig}} - L_{\mathrm{Aug}})/L_{\mathrm{Orig}}$

domain-planner-macro	+S -s	+T -t	P ± p	+L -l	Q ± q
NCokeCake-FF-1	+28 -0	+72 -0	82 ± 1	+62 -10	4 ± 0
NCokeCake-FF-2	+28 -0	+72 -0	82 ± 1	+60 -12	3 ± 0
NCokeCake-SGPlan-1	+36 -4	+28 -10	-1 ± 41	+12 -26	-6 ± 2
NCokeCake-SGPlan-2	+44 -4	+32 -6	55 ± 17	+0 -38	-26 ± 2
NRailway-FF-1	+24 -0	+36 -12	58 ± 7	+2 -34	-12 ± 2
NRailway-FF-2	+20 -2	+36 -10	51 ± 8	+0 -34	-14 ± 2
NRailway-SGPlan-1	+16-0	+60 -14	1 ± 34	+2 -62	-18 ± 2
NRailway-SGPlan-2	+16-0	+60 -16	1 ± 34	+2 -62	-16 ± 2
NReliefWork-FF-1	+30 -0	+70 -0	99 ± 0	+16 -44	-1 ± 0
NReliefWork-FF-2	+30 -0	+70 -0	99 ± 0	+42 -6	2 ± 0
NReliefWork-SGPlan-1	+10 -0	+80 -0	88±1	+0-20	0±0
NReliefWork-SGPlan-2	+10 -0	+80 -0	88±0	+0-20	0±0
PRailWay-FD-1	+50-0	+20-2	41 ± 18	+18 -2	13 ± 3
PRailWay-FD-2	+54-0	+20-2	52 ± 8	+16 -6	7 ± 3
PRailWay-FF-1	+14 -4	+22 -4	10 ± 23	+0 -18	-5 ± 1
PRailWay-FF-2	+12 -0	+26 -4	21 ± 8	+0 -12	-10 ± 3
PRailWay-SGPlan-1	+14 -0	+50 -12	34 ± 7	+24 -34	0 ± 0
PRailWay-SGPlan-2	+14 -0	+56 -6	41 ± 7	+2 -60	-4 ± 0
PReliefWork-FD-1	+0 -0	+80 -20	7 ± 3	+20 -36	0 ± 0
PReliefWork-FD-2	+0 -0	+88 -12	10 ± 2	+16 -42	0 ± 0
PReliefWork-FF-1	+0 -0	+100 -0	89 ± 0	+0 -100	-3 ± 0
PReliefWork-FF-2	+0 -0	+100 -0	89 ± 0	+2 -54	0 ± 0
PReliefWork-SGPlan-1	+20 -0	+54 -8	44 ± 5	+0-12	0 ± 0
PReliefWork-SGPlan-2	+36 -0	+62 -0	75 ± 2	+0-60	0 ± 0
PReliefWork-VHPOP-1	+16 -0	+24 -0	92 ± 2	+0 -2	0 ± 0
PReliefWork-VHPOP-2	+12 -0	+24 -0	91 ± 2	+0 -2	0 ± 0
VHPOP, FD does not support FLUENTS; VHPOP cannot solve any PRailway problem					

Figure 7: Summarised experimental results

Justification: Our method uses a rich collection of genetic operators to explore a wider macro space that includes both observable and non-observable macros. Its implementation does not use any planner or domain specific knowledge anywhere in macro representation, generation, validation, and evaluation. Figure 7 shows that, for most planner-domain pairs, using our non-observable macros, not only can problems be solved much faster but also many unsolvable problems can be solved. Figure 8 shows how observable macros are transformed into non-observable macros; the non-observable macro (the middle one in the figure) was not observed from the examples used although it could be found in plans of other problems. ■

Hypothesis 3 *Non-observable macros, as suggested by our method, could help improve planners' performances on domains significantly and in many cases they are suggested over or better than observable ones.*

Justification: This is the main result of this paper as it shows non-observable macros are worth exploring. Results present in Figure 7 shows such macros in most cases help planners run faster on domains. As mentioned earlier all these macros are suggested over observable macros; as an example, the last chart in Figure 8 shows performance comparison of the evolved-best and the observed-best macros in PSatellite for VHPOP. ■

Hypothesis 4 *There exist catalytic macros that are not usable in plans but still help speedup search.*

Justification: This is a very interesting result from our experiments. Catalytic macros, according to our observation so far, have unsatisfiable preconditions and incoherent action sequences. Although they cannot be used in plans, they somehow help speedup search. Our investigation shows this is not necessarily due to improvements in the heuristic values. Heuristic functions normally involve relaxed problem formulations, which are inconsistent with respect to the given formulations. We remark catalytic macros, being inconsistent themselves, are somehow helpful in the heuristic evaluations. In consequence, successor node selection is affected; the search is taken in a different direction; and the goal is found quickly. Figure 8 shows the first action in the catalytic macro makes it non-usable in any plans; however, FF still runs faster (even compared to the other macros in the figure) using this macro in Blocks. There exist examples for other planners and domains as well. ■

		(drop ?b5)
(unstack ?b5 ?b0)	(unstack ?b5 ?b0)	(unstack ?b5 ?b0)
(stack ?b5 ?b2)	(stack ?b5 ?b2)	(stack ?b5 ?b2)
(pick ?b0)	(pick ?b0)	(pick ?b0)
	(stack ?b0 ?b5)	(stack ?b0 ?b5)

Figure 8: Evolution of macros in Blocks for FF: observable to non-observable to catalytic non-observable

Other observations and comments about our experiments are as follows:

1. Our non-observable macros have mixed effect on plan length (see +L/-l in Figure 7).
2. In essence, this work is to show that such an evolutionary approach works successfully; its performance is, therefore, not measured in terms of its learning time. However, Figure 9 depicts learning effort required for planners on domains in this work. To speed up the learning process, the intuitively chosen parameters (*e.g.* population-size, epoch-count, operator-probabilities, replacement-level, etc.) are to be tuned.
3. Composition of actions is not, in essence, a hard requirement of this work; had PDDL supported macros, a macro could be executed or constructed online easily.

T: training hours, N: #macros evaluated, R: ratio of #generated to #evaluated

T / N / R	FF	SGPLAN	FD	VHPOP
NCokeCake	77 / 1380 / 2	100 / 1460 / 3		
NRailway	232 / 2560 / 2	209 / 3620 / 3		
NReliefwork	66 / 3624 / 7	83 / 4272 / 10		
PRailway	233 / 2840 / 3	140 / 4000 / 8	90 / 4000 / 6	
PRailway	238 / 2208 / 6	121 / 3312 / 8	129 / 2160 / 3	142 / 1656 / 5

Figure 9: Learning effort: training time, macros evaluated, and the ratio of macros generated to evaluated

Conclusion

This paper presents an offline evolutionary macro learning method that works with arbitrary planners and domains. Most existing work, exploiting specific features of planners or domains, learns macros that are observable from examples or from domain analysis. But, the examples might not cover many aspects of the system, or might not always reflect that better choices have been made during the search. Macros learnt from domains are mainly coherently connected or abstraction based; also they cannot capture properties of a planner in use. Nevertheless, most of these methods explore restricted macro spaces. Our method, in contrast, explores a wider macro space and learns macros that are somehow not observable from the examples. Moreover, the learning method does not discover or utilise any specific structural properties of planners or domains. We have achieved a convincing, and in many cases dramatic, improvement with a number of planners and several domains. In summary, this paper successfully demonstrates that the non-observable macros, even the catalytic ones, help achieve significant improvement in a planner's performance; so, any intuitive restrictions on the macro space, as imposed by existing methods, could be loosing much potential of macros in planning. As we consider only individual macros for the time being, we hope to extend our approach to learning a set of macros either incrementally or using a genetic approach on macro-sets. In the latter case, the challenge is to explore both macro space and macro-set space together.

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.

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. 2007. MARVIN: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

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.

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*.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planner and domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

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*.