

Learning Macro-Actions for Arbitrary Planners and Domains

M.A. Hakim Newton and John Levine and Maria Fox and Derek Long

Computer and Information Sciences

University of Strathclyde

Glasgow, United Kingdom

e-mail: {newton, johnl, maria, derek}@cis.strath.ac.uk

Abstract

Many complex domains and even larger problems in simple domains remain challenging in spite of the recent progress in planning. Besides developing and improving planning technologies, re-engineering a domain by utilising acquired knowledge opens up a potential avenue for further research. Moreover, macro-actions, when added to the domain as additional actions, provide 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 work on macros exploits properties explicitly specific to the planners or the domains. However, such properties are not likely to be common with arbitrary planners or domains. Therefore, a macro learning method that does not exploit any structural knowledge about planners or domains explicitly is of immense interest. This paper presents an offline macro learning method that works with arbitrarily chosen planners and domains. Given a planner, a domain, and a number of example problems, the learning method generates macros from plans of some of the given problems under the guidance of a genetic algorithm. It represents macros like regular actions, evaluates them individually by solving the remaining given problems, and suggests individual macros that are to be added to the domain permanently. Genetic algorithms are automatic learning methods that can capture inherent features of a system using no explicit knowledge about it. Our method thus does not strive to discover or utilise any structural properties specific to a planner or a domain.

Introduction

Planning has achieved significant progress in recent years from planning competitions. However, the focus of planning research remains mostly on developing and improving planning technologies. Diverse planning architectures are being put forward; structural knowledge about search algorithms or problem instances are being incorporated into planners; even other technologies (*e.g.* Satisfiability, Model Checking etc.) are being translated into planning. But 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 to achieve such domain enhancement.

Copyright © 2007, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

A *macro-action*, or *macro* in short, is a group of actions selected for application at one time like a single action. Macros could represent high level tasks comprising low level details. From a broader perspective, macros are like *subroutines* or *procedures* in the *programming paradigm*. However, macros are 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. Thus, macros could capture local search in the troublesome regions of the search space and encapsulate significant experience of the planner. Consequently, a goal could be *reached* quickly and problems that are *unsolvable*¹ could become *solvable*.

Contribution

Most existing macro learning methods are more focused and specialised to exploiting particular planner or domain properties. For example, Macro Problem Solver (MPS) (Korf 1985) learns macros for a particular goal in domains that exhibit operator decomposability; MARVIN (Coles & Smith 2004) learns macros that help a forward chaining heuristic based planner escape plateaus in its heuristic profile (for other examples, see the section discussing related work). However, such properties are not likely to be common with a wider range of planners or domains. Therefore, a macro learning method that does not exploit any explicitly specific structural knowledge about planners or domains remains unexplored. This paper presents an offline method that learns macros genetically from plans for arbitrarily chosen planners and domains. The macros are generated from plans of *smaller problems*² and evaluated against other *larger* but solvable problems. This is to show that macros learnt from smaller problems can reliably be applied in larger problems. The generality aspects of our method, however, are due to the use of a genetic algorithm as our learning technique and

¹By *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*.

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

plans as the macro generation source. On one hand, genetic algorithms are automatic learning methods that can capture inherent features of a system (*e.g.* what is good or bad of it) using no explicit knowledge about it. Plans, on the other hand, invariably reflect successful choices of actions by the planner to cross the problem state spaces. Also, plans could inherently bear the characteristics of the planner or the domain, especially that led to the solutions. Our method thus does not discover or utilise any knowledge explicitly specific to a planner or a domain.

Given a planner, a domain, and a number of example problems, our method learns macros from plans under guidance from a genetic algorithm. It then suggests individual macros that are to be added permanently to the domain as additional actions. For the sake of convenience, 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 the smaller example problems to seed the population. To explore only 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 into two, 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 but solvable problems (the remaining examples) with the macro augmented domains and the original domain. After the learning is accomplished, yet another set of more difficult problems (which might include unsolvable instances) are used to demonstrate the performance of the selected individual macros. We have achieved convincing results with several planners and domains.

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

Motivations

Conceptually a system achieves better performance if it can exploit its previous experiences. Our highest level objective is to learn experiences of a system in certain contexts and to provide them somehow to the system. Also, the knowledge acquired from simpler situations should reliably be applicable to more complex situations. Moreover, we would like to achieve generality of our learning method over the systems for which it learns, over the knowledge it acquires for them, and over the way knowledge is conveyed.

Learning from Examples From the learning perspective, it is very important that knowledge be acquired from simpler situations, reinforced in complex but manageable situations, and applied in yet more complex and even unmanageable situations. The success of the first two activities depends on the achievement of performance and manageability in the last activity. Macros are, therefore, generated from plans of smaller problems and evaluated against other larger but solvable problems. To demonstrate the performance of the sug-

gested macros, yet larger problems are used, which might include unsolvable instances.

Learning in Planning From previous research, it has become obvious that planning in any realistic domain requires much knowledge. Systems that exploit particular domain or planner aspects have demonstrated success. But they 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 method that works unconditionally meaning irrespective of any particular characteristics exhibited by the planner or the domain.

Macros as Knowledge Conveyors The knowledge acquired from a particular context can be incorporated into the planner or encoded into the domain. The first approach taken by most existing work needs extension of the planner. The second approach however does not need that, if macro-actions are used and represented like normal actions. When macros are added into a domain as additional actions, the *reachability* of a problem is not affected. But they cause more pre-processing 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. Within syntactical and semantical limits of the Planning Domain Definition Language (PDDL), knowledge modelled as obligations (*i.e.* control rules) is not supported and any knowledge can be conveyed only by additional choices (*i.e.* actions). Macro-actions in the form of normal actions are thus convenient to achieve domain enhancement.

Macros from Plans Plans invariably reflect the successful choices of actions to cross the problem state space and thus could bear the characteristics of the planner, the domain or the problem inherently. For example, 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 the 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. Our work, therefore, explores only the macros that occur in plans.

Learning Macros Genetically

Although the macro space is restricted when macros are learnt only from plans, an exhaustive approach is not good because macros comprising any number of actions are to be considered. This work takes guidance from a genetic algorithm while searching the macro space.

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 until the termination condition is met. In a particular problem context, an individual is taken for a so-

lution (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. 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.

Genetic Algorithms in Planning Genetic 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. Earlier, Spector managed to achieve plans for a range of initial and goal states (Spector 1994), but the problems were very small in size. SINERGY (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 it is somewhat inferior to the state-of-the-art planners. Genetic algorithms have also been used to optimise plans (Westerberg & Levine 2001).

Related Work

Macros are not very new in planning research. Therefore, it is useful to compare our approach with other previous work.

STRIPS (Fikes, Hart, & Nilsson 1972) produces its macros from all unique subsequences of wholly parameterised plans. The number of macros thus grows quickly. Our method, in contrast, learns and suggests the best individual macros for any given planner-domain pair.

REFLECT (Dawson & Siklóssy 1977) generates macros in a preprocessing stage by analysing possible causal links between actions in a domain. This approach largely depends on domain characteristics and ignores how macros would impact on planners. Also, this approach does not consider macros that have concurrent actions or that help a particular planner syntactically. Our work does not consider such stringent restrictions and our macros are tested with the planner during their evaluation.

MORRIS (Minton 1985) performs exhaustive search on plans to learn macros for STRIPS from plan fragments that are frequently used or achieve interactive goals. Our method, in contrast, uses genetically guided search on the plan fragments and does not assume any structure being present in the domain.

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 on domains that exhibit operator decomposability. MPS needs a different set of macros when the problem instances scale or goals are

different. Unlike our method, MPS therefore relies on the presence of specific characteristics in the domains and the problems.

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 usage of macros in plans. Unlike our method, this approach therefore depends on particular characteristics of the search algorithm and the domain. However, like our method, MACLEARN compiles macros into regular actions and adds them into the domain.

MARVIN (Coles & Smith 2004) generates macros from the plan of a reduced version of the given problem after eliminating symmetries. MARVIN also learns macros from action sequences that lead its FF style search to successfully escape plateaus (which is a planner characteristic). Our method, in contrast, is not aware of any such properties be present in a planner. In particular, we can learn macros that eliminate such plateaus from FF's search space.

Macro-FF (Botea *et al.* 2005), an extension of FF to incorporate macros, uses component level abstraction based on static facts of a domain to learn macros. It also lifts partial-order macros from plans based on an analysis of causal links between successive actions. However, our work does not exploit any specific domain characteristics, or a stringent constraint like causal links between constituent actions. Although both methods evaluate macros by solving problems, Macro-FF considers improvement in the number of states explored whereas our method uses time gains. This is because improvement in states explored does not necessarily translate into time efficiency as evaluation or exploration of each state might take more time when macros are used.

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. However, both of these methods exploit domain characteristics.

A Macro Learning Method

Our learning method is described in Figure 1. This is based on a genetic approach with individuals being taken as macros. Its implementation issues include representation, generation, and evaluation of macros along with validation and pruning techniques to reduce possible wasted effort.

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: A learning method using a genetic approach

Macro Representation

Genetic algorithms require individuals to be encoded in a composite form whereas this work requires macros to be added 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. Had PDDL supported macros syntactically, the action composition would not be required and planners could easily execute a macro sequence.

```

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

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). Not every composition produces 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 work considers composition of actions only in STRIPS and FLUENTS subsets of the PDDL.

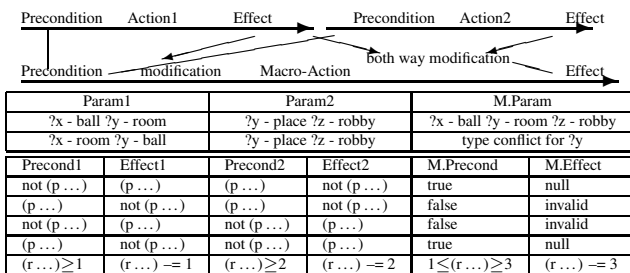


Figure 3: A composition of actions by regression

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 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).

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; distinct variables however remain unaffected.

Example Problems

The example problems, our method requires, are to be supplied as input. Alternatively, a problem generator can be provided, which is used to generate the necessary problems randomly. A set of smaller problems called *seeding problems* are solved and the plans are used as the macro generation source. Another set of larger but solvable problems called *ranking problems* are used for macro evaluation. Note, in order for the time gain to be measurable significantly and precisely on a given computer, the ranking problems cannot be very small. For this work, the ranking problems are solvable in 10secs as shown in Figure 7.

Although, for the time being, we use randomly generated problems, it is worth mentioning here that the selection of problems normally affects the suitability of a macro in covering a wider range of problems. Carefully chosen examples should cover as many aspects of the system as possible.

Macro Generation

Generation of macros require genetic operators to be defined. One motivation of this work is to generate macros from plans. Besides, our observations suggest subsequences of a good sequence are also good while sequences containing bad subsequences are also bad. Genetic operators are therefore 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). Lifting of random sequences from plans is used as yet another operator which facilitates diversity of the macro space exploration. To seed the initial population, sequences of actions are randomly lifted (using the *lift* operator) from plans of the seeding problems.

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

Macro Evaluation

To evaluate individual macros, a number of different problems called *ranking problems* are used. These problems are larger than the seeding problems, but not too large because they are attempted to be solved for every macro. 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. Although a *deterministic* planner takes the same time and returns the same plan every time a problem is solved, a *stochastic* planner takes varying times and returns different plans. Assuming the underlying time distributions to be *normal* for a stochastic planner, a problem is therefore solved a number of times and a random variable having parameters (sample-count ν , mean μ , dispersion $\delta = \sigma/\sqrt{\nu}$) is used to represent the time distribution. Based on relevant fitness criteria, the augmented domain (and so the macro) is then given a numerical rating against the original domain.

Fitness Criteria For a good macro, in qualitative terms, *most problems* should be solved taking *less time* in *most cases* with its augmented domain. A bad macro, in contrast, would cause overhead that leads to longer solution times or even failures in solving problems within given resource limits. A good macro, however, may not have *high usage* because there could be an infrequently used but *tricky* macro that saves enormous search time. Furthermore, good macros need not be intuitively natural. For a given macro, we therefore require three quantitative measures.

Cover (C) measures the portion of the ranking problems solved using its augmented domain. Note, all problems are solvable.

Score (S) measures the weighted mean time gain/loss over all the ranking problems compared to when they are solved using the original domain. Any gain/loss for a larger problem gets more weight as our interest is to apply macros in larger problems.

Point (P) measures the portion of the ranking problems solved with the augmented domain taking less or equal time compared to when they are solved using the original domain.

Utility Value Reflecting all the three fitness factors together, the formulae shown in Figure 5 gives a numerical rating (also called *fitness value*) to a macro. Among the three factors, score is more effective in the ranking of good macros while the other two are to counterbalance any misleadingly high utility value. The individual factors are calculated in slightly different ways for deterministic and stochastic planners. However, Figure 5 shows a unified formula considering (only for convenience of description) deterministic planners as a special case of stochastic planners. Notice that, most calculated values are normalised in $[0,1]$. The notion used in computation of s_k and s'_k will be clear from their values at certain points (e.g., $s_k = 1, \frac{1}{2}$, and 0 for $\mu'_k = 0, \mu_k$, and ∞ respectively). Moreover, its non-linear characteristic is suitable for a utility function. Note, the utility values assigned to the macros are not absolute in any sense, rather relative to the ranking problems and the planner used.

Macro Validation

Macros having unsatisfiable preconditions or inconsistent effects are detected whenever possible in Step 2(a)i of the learning method in Figure 1. Unsatisfiable preconditions, however, cannot be detected completely at this stage (note that, *satisfiability* is also a research problem). Besides, inconsistent effects sometimes arise during the runtime of a planner. The reason is mostly the mishandling of *parameter binding* and *object inequality* by some planners especially when more than one parameters have compatible types. For

$$U = \begin{cases} C \times S \times P \\ = -\frac{1}{2} \text{ if } C = 0 \\ = -1 \text{ if invalid plans produced} \end{cases} \quad \left| \begin{array}{l} C = \sum_{k=1}^n c_k / n \\ S = w \sum_{k=1}^n s_k w_k + w' \sum_{k=1}^n s'_k w'_k \\ P = \sum_{k=1}^n p_k \end{array} \right.$$

Where,

n : Number of ranking problems to be solved.

m : Number of times a ranking problem is to be solved. For a deterministic planner, $m = 1$.

$t_k(\nu_k, \mu_k, \delta_k)$: 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 $\delta'_k = 0$.

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

$c_k = \nu'_k / \nu_k$: 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.

$s'_k = \delta_k / (\delta_k + \delta'_k)$: 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 $\delta_k = \delta'_k = 0$.

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

$w'_k = 1/n$: Weight of gain/loss in dispersion with 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 Student's 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 δ s are zero.

Figure 5: A utility function for macro evaluation

example, if both $?x$ and $?y$ are bound with the same object in $(p ?x)$ and $(\text{not } (p ?y))$, the instantiated effect becomes inconsistent and causes invalid plans to be generated. Appropriate not-equalities are, therefore, added to the precondition whenever parameters having compatible types represent different objects. Moreover, in many unfamiliar cases, planners produce invalid plans due to some unknown reasons (probably bugs). Plans produced with the augmented domains are, therefore, validated for such planners in Step 2(a)ii.

Macro Pruning

Pruning techniques, several comes from existing work, are used to reduce any effort wasted otherwise to explore potentially inferior macros.

Pruning during generation: The following strategies are adopted in Step 2(a)i of the learning method in Figure 1:

1. Sequences of actions that have subsequences producing null effects are not minimal.
2. The more the parameters, the more the unnecessary instantiated operators. This affects the planners which do operator instantiation in their preprocessing steps.
3. Longer action sequences are more specific to certain objectives and are less likely to be useful for a wider range of problems.
4. Similar sequences of actions differing only by parameterisation are considered copies of a single sequence.

5. Sequences of actions equivalent in partial order are considered copies of a single sequence.
6. Actions in a macro should have parameters in common (by name). This ensures cohesiveness of the constituent actions and also oversees that irrelevant actions are not part of a macro. A more stringent strategy that consecutive actions in a macro must have a causal link between them has been considered but finally not 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: Failure to solve a problem using the augmented domain within certain limits whereas it is solvable using the original domain implies the macro causes much overhead and resource (time, memory, etc.) scarcity to the planner. Early detection of such inferior macros in Step 2(a)ii in Figure 1 saves learning time needed otherwise to solve the remaining problems.

Experiments

To demonstrate that this learning method works for arbitrary planners, we choose a number of planners (see Figure 6) from different tracks (*i.e.* planning styles), but only those holding the basic characteristics of the tracks. Moreover, to show the effectiveness of our macros against planning technologies, most of the planners chosen are the current state-of-the-art ones in their respective tracks. Similarly, the domains chosen (see Figure 6) are bench mark domains used in planning research. The problems used to demonstrate performance of the suggested macros are called *testing problems*. These are larger than the ranking problems and might include unsolvable instances. For a suggested macro, the testing problems are solved using both the original domain and the augmented domain.

- ◊ **FD** (Fast Downward) is a heuristic based progression planner that uses a multi-valued encoding and a hierarchical problem decomposition technique to compute its heuristic function.
- ◊ **FF** (Fast Forward) is a forward chaining heuristic based state space planner that uses a relaxed *graphplan* algorithm as its heuristic.
- ◊ **LPG** (Local Search for Planning Graphs) is a stochastic planner that uses a heuristic based efficient local search on *action graphs* representing partial plans.
- ◊ **SatPlan** transforms a planning problem into SAT-instances, solves them using SAT-solvers, and the solutions are transformed back to give a plan.
- ◊ **SGPlan** partitions a large planning problem into subproblems, solves them by some other planner, and the plans are combined to produce a global solution.
- ◊ **VHPOP** (Versatile Heuristic Partial Order Planner) is a partial order causal link planner that uses various *flaw selection strategies* as its heuristic.
- ◊ **Blocksworld** domain has a robot arm that picks and drops blocks to build stacks on a table.
- ◊ **Ferry** domain requires a number of cars to be transported between ports by a ferry. The ferry can carry only one car at a time.
- ◊ **Gripper** domain has a robot with two grippers to carry balls between two rooms.
- ◊ **Satellite** domain deals with a number of satellites that can take images of targets in various modes and transmit data to the base.
- ◊ **NFerry** is a numeric version of the Ferry domain described above with additional numeric constraints on fuel that is consumed by the ferry while sailing.
- ◊ **NSatellite** is a numeric version of the Satellite domain described above with additional numeric constraints on buffer capacity.

Figure 6: Planners and domains used in this work

Results

Figure 7 describes the typical setup of our experiment. The parameter values in most cases are chosen intuitively. Figure 8 summarises the performance of the suggested macros for the planners. Moreover, Figure 9 optionally gives a graphical illustration of plan times for some of the macros.

- * Number of random problems: Seeding 5, Ranking 20, Testing 50
- * Macro size limits: Maximum parameters 8, Maximum sequence length 8
- * Operator selection probability: Extend 25%, Shrink 25%, Split 25%, Lift 25%
- * 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: $2 \times$ number of actions
- * Satisfactory replacement level: at least 1 in every 25 consecutive epochs
- * Generation attempts: maximum 999999 for every new macro
- * Resource limit: memory 1 gigabyte, time - ranking 10 secs, testing 1800 secs

Figure 7: Experimental setup

- $S\%$ 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 $(T_{Orig} - T_{Aug})/T_{Orig}$
- $(Q\%, q\%)$ is (mean, dispersion) of plan length (L) quality $(L_{Orig} - L_{Aug})/L_{Orig}$

domain-planner-macro	+S -s	+T -t	P \pm p	+L -l	Q \pm q
Blocks-FF-1	+36 -0	+60 -4	65 \pm 19	+58 -4	20 \pm 2
Blocks-FF-2	+26 -4	+40 -16	18 \pm 14	+20 -36	-4 \pm 2
Blocks-LPG-1	+0 -0	+94 -0	58 \pm 2	+92 -0	28 \pm 2
Blocks-LPG-2	+0 -0	+74 -0	41 \pm 3	+80 -0	19 \pm 1
Blocks-SGPlan-1	+6 -0	+72 -12	-19 \pm 76	+38 -34	-2 \pm 2
Blocks-SGPlan-2	+8 -0	+42 -48	-60 \pm 28	+12 -74	-18 \pm 2
Blocks-VHPOP-1	+54 -0	+26 -2	79 \pm 9	+0 -2	-2 \pm 2
Blocks-FD, SatPlan	No good macro could be learnt				
Ferry-FD-1	+0 -0	+100 -0	93 \pm 0	+96 -4	9 \pm 0
Ferry-FF-1	+0 -0	+100 -0	92 \pm 0	+0 -100	-30 \pm 0
Ferry-FF-2	+0 -0	+100 -0	92 \pm 0	+0 -100	-30 \pm 0
Ferry-LPG-1	+0 -0	+92 -0	92 \pm 0	+0 -92	-18 \pm 0
Ferry-LPG-2	+0 -0	+92 -0	90 \pm 0	+0 -92	-19 \pm 0
Ferry-SatPlan-1	+10 -0	+88 -2	94 \pm 2	+0 -90	-64 \pm 5
Ferry-SatPlan-2	+6 -4	+74 -12	-2 \pm 36	+0 -86	-54 \pm 1
Ferry-VHPOP-1	+68 -0	+32 -0	100 \pm 0	+0 -32	-26 \pm 2
Ferry-VHPOP-2	+62 -0	+32 -0	100 \pm 0	+0 -4	-1 \pm 1
Ferry-SGPlan	No good macro could be learnt				
Gripper-FD-1	+54 -0	+46 -0	88 \pm 1	+0 -46	0 \pm 0
Gripper-FD-2	+54 -0	+46 -0	87 \pm 1	+0 -46	0 \pm 0
Gripper-FF-1	+0 -0	+100 -0	97 \pm 0	+0 -100	-31 \pm 0
Gripper-FF-2	+0 -0	+100 -0	97 \pm 0	+0 -100	-31 \pm 0
Gripper-LPG-1	+0 -0	+100 -0	92 \pm 0	+0 -100	-23 \pm 0
Gripper-LPG-2	+0 -0	+100 -0	90 \pm 0	+0 -100	-21 \pm 0
Gripper-SatPlan-1	+20 -0	+78 -2	77 \pm 4	+0 -80	-47 \pm 1
Gripper-VHPOP-1	+36 -0	+64 -0	85 \pm 4	+0 -36	-17 \pm 3
Gripper-VHPOP-2	+34 -0	+64 -0	99 \pm 0	+0 -34	-11 \pm 2
Gripper-SGPlan	No good macro could be learnt				
Satellite-FF-1	+0 -0	+100 -0	96 \pm 0	+0 -100	-43 \pm 1
Satellite-FF-2	+0 -0	+100 -0	94 \pm 0	+0 -100	-43 \pm 1
Satellite-VHPOP-1	+12 -0	+22 -6	51 \pm 21	+0 -28	-13 \pm 2
Satellite-VHPOP-2	+14 -0	+24 -6	47 \pm 21	+0 -42	-21 \pm 2
Satellite-FD, LPG, SatPlan, SGPlan	No good macro could be learnt				
NFerry-FF-1	+32 -0	+62 -6	66 \pm 6	+0 -68	-85 \pm 3
NFerry-FF-2	+12 -42	+22 -4	50 \pm 9	+0 -26	-87 \pm 4
NFerry-SGPlan-1	+4 -0	+46 -36	33 \pm 8	+14 -68	-8 \pm 1
NFerry-SGPlan-2	+18 -0	+44 -38	33 \pm 10	+0 -82	-36 \pm 2
NFerry-LPG	No problem could be solved by the planner				
NFerry-FD, SatPlan, VHPOP	FLUENTS not supported by the planners				
NSatellite-FF-1	+58 -0	+42 -0	98 \pm 0	+0 -42	-10 \pm 1
NSatellite-FF-2	+58 -0	+42 -0	97 \pm 0	+4 -38	-9 \pm 1
NSatellite-LPG-1	+64 -0	+38 -0	48 \pm 3	+0 -18	-11 \pm 1
NSatellite-LPG-2	+64 -0	+38 -0	48 \pm 3	+2 -16	-10 \pm 1
NSatellite-SGPlan-1	+0 -0	+100 -0	39 \pm 1	+0 -100	-29 \pm 1
NSatellite-SGPlan-2	+0 -0	+100 -0	28 \pm 0	+0 -100	-28 \pm 1
NSatellite-FD, SatPlan, VHPOP	FLUENTS not supported by the planners				

Figure 8: Summarised experimental results

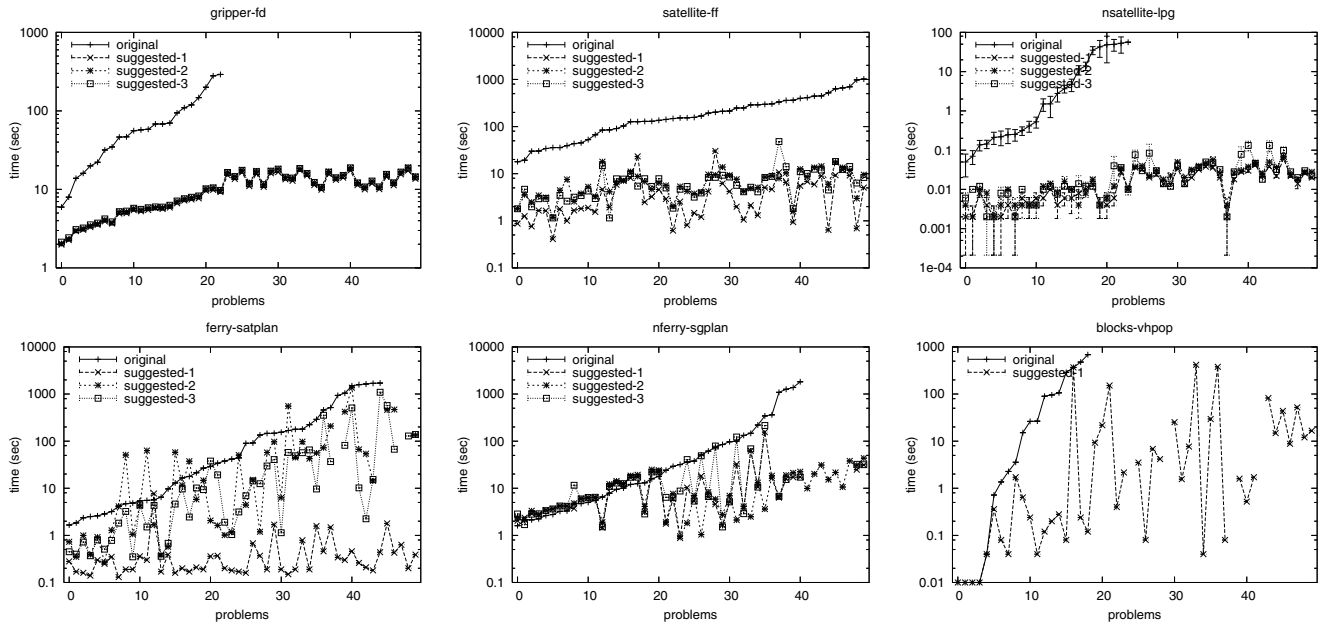


Figure 9: Time performance of some suggested macros against original domains

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 example 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 found to be consistent and positively correlated with the values assigned against ranking problems during evaluation. Furthermore, the suggested macros, in many cases, (see Figure 8) achieve significant improvement with the testing problems. This implies macros learnt by our method from the smaller problems are equally useful for the larger problems. ■

Hypothesis 2 *Our method can learn macros from plans effectively without exploiting any knowledge explicitly specific to planners, domains, problems, or plans.*

Justification: Our method does not discover or utilise any explicitly specific properties from either of planners, domains, problems or plans. Moreover, it only explores the macros that occur in plans. Figure 8 shows that, we can learn useful macros for most planner-domain pairs. ■

Hypothesis 3 *Macros learnt by our method can lead to significant improvement of planners' performance on domains.*

Justification: In spite of much work on macros, it is not known whether one macro learning technique can effectively be used for arbitrary planners and domains. This work

shows (see Figure 8), using our macros not only can problems be solved much faster with different planners on different domains but also many unsolvable problems can be solved. ■

Hypothesis 4 *The macros learnt by our method can capture various features of domains and planners.*

Justification: In some domains, the best macros learnt for different planners are found to be overlapping. This suggests such macros might be domain specific or in other words planner independent. Besides, using our macros, FF suddenly finds its search space much more friendly (*i.e.* relatively less plateaus) and LPG does not require so many restarts in its search. Also, in the problems that are not easy to partition, SGPlan solves them faster with our macros. However, the opposite happens with easily decomposable problems as seen with Blocks and NFerry domains. We do not know as yet how other planners get help from our macros. Nevertheless, our method thus learns macros that capture various characteristics of domains and planners. ■

Other observations and comments about our experiments are as follows:

1. Our macros often lead to longer plans as observed with most domains and planners.
2. The evaluation of a macro takes much more time than the generation of a new macro even though many pruning techniques are used in both phases. To speed up the learning process, the intuitively chosen parameters (*e.g.* population-size, epoch-count, replacement-level, operator-probabilities, etc.) are to be tuned. However, in essence, this work is to show that such a generalised approach works successfully; its performance is, therefore, not measured in terms of its learning time (see Figure 10 for a brief illustration).

Time(Hours)	fd	ff	lpg	satplan	sgplan	vhpop
blocks	14.5	19.5	58.7	3.1	9.1	0.5
ferry	17.2	15.4	32.2	1.0	19.1	0.3
gripper	4.1	3.1	21.2	1.1	3.0	0.7
satellite	31.2	24.3	152.4	0.8	13.7	1.2
nferry		23.0			35.6	
nsatellite		38.4	30.0		27.0	

Figure 10: Macro learning time for planners on domains

- Our work could be extended to temporal domains had PDDL been closed under composition with *durative actions* or did it support discrete effects at any time over the duration of a durative-action. Nevertheless, composition of actions is not, in essence, a hard requirement of this work; had PDDL supported macros, the actions of a macro could be executed easily.

Conclusion

This paper presents an automated macro learning method that requires no structural knowledge about the domains or the planners. Despite recent significant progress in planning, many complex domains and larger problems in simple domains remain challenging. However, the focus of planning research remains mostly on developing and improving planning technologies. Also, the impact of problem formulation on its solution process remains overlooked. Macros provide a promising avenue in planning research to achieve further improvement through domain enhancement. 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 be encoded as macros while re-engineering a domain. Most existing work on macros somehow need knowledge specific to the planner or the domain. A macro learning method that does not need such knowledge is, therefore, important. Our method learns macros effectively from plans for arbitrarily chosen planners and domains 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 a convincing, and in many cases dramatic, improvement with a number of planners and several domains. Also, we have demonstrated successfully by our results that macros learnt from smaller problems can reliably be applied in larger problems. Further experiments to learn macros for more complex domains are underway. 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. It would be useful to find what problems are suitable for learning as we use randomly generated problems for the time-being. However, this is a common issue for any such machine learning approaches. Although one motivation behind this work is to capture a planner's experiences on a domain landscape, we are also motivated by the desire to investigate how evolution (a genetic algorithm with richer collection of operators) of such knowledge further improves its performance.

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*.
- 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.
- 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.
- 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*.
- 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*.