Implicit Learning of Compiled Macro-Actions for Planning

M.A. Hakim Newton[1] and **John Levine**[2]

Abstract. We build a comprehensive macro-learning system and contribute in three different dimensions that have previously not been addressed adequately. Firstly, we learn macro-sets considering implicitly the interactions between constituent macros. Secondly, we effectively learn macros that are not found in given example plans. Lastly, we improve or reduce degradation of plan-length when macros are used; note, our main objective is to achieve fast planning. Our macro-learning system significantly outperforms a very recent macro-learning method both in solution speed and plan length.

1 INTRODUCTION

Macros provide one potential avenue for knowledge-assisted planning. Macro-learning systems performed promisingly in the International Planning Competition (IPC) 2008 Learning Track. *Macros* are sequences of actions applied at one time like single actions. Macros have been learnt from reusable plan fragments [7], particularly that are frequently used [6] or causally linked [1] or solve difficult subproblems [6]. They have also been captured from action sequences that help find better successor nodes on the search space, particularly when the immediate search neighbourhood is not good [2, 4, 5].

In this work, we mainly build a comprehensive macro-learning system and contribute in three different dimensions that have previously not been addressed adequately.

- **Macro-Set Learning.** A number of macros, when used together during planning, interact with each other in interesting ways. Therefore, a set of individually best-performing macros does not necessarily perform the best. Although macros can help reach a goal with fewer intermediate states explored, they increase the branching factor at every choice point. Considering these, we implicitly learn macro-sets based on their aggregate performances.
- **Unobserved Macro Learning.** A given set of example plans might not contain many action sequences (called unobserved macros) that are useful in many other problem instances in the domain. However, many unobserved macros could still be constructed using the actions from the given plans. In this work, we effectively learn such macros that are not found in *given* example plans but achieve comparable performances.
- **Plan Length Consideration.** Macros, when replaced by constituent actions, often result in longer plans than when no macro is used; in the Gripper domain, a macro to carry only one ball (rather than two) at a time results in many more-than-necessary move actions in the plan. In this work, we improve or reduce degradation of plan-length of the plans produced with macros. Note that our main objective is to achieve fast planning.

[1] National ICT Australia (NICTA) and IIIS, Griffith University, Australia. E-mail: mahnewton@nicta.com.au

[2] Computer and Information Sciences, University of Strathclyde, United Kingdom, E-mail: johnl@cis.strath.ac.uk

We view macro-learning as a search problem on the macro and macro-set spaces. The macro space is bounded by parameter-count and action-count, and the macro-set space by macro-count. Macros are constructed using actions lifted from given generalised plans. However, they include both observed and unobserved macros. Macro-sets are constructed using macros that have a certain minimum performance level. The performance is measured both in solution speed and plan length. We explore the search spaces by using a thoughtfully-designed population-based stochastic local search algorithm. The search algorithm is similar to genetic algorithms although there exist significant dissimilarities.

In our implementation, we consider the skeleton of an algorithm used in [7] as a search framework (henceforth referred to as the framework). Our contribution is not in the search algorithm, rather in i) *extending the neighbourhood functions* to generate unobserved macros, ii) *reworking the fitness function* to include plan-length measures, and iii) *perform macro-set exploration* after learning individual macros. Using benchmark planning domains in the STRIPS subset of PDDL and a state-of-the-art planner FF [3], our macro-learning system achieves convincing empirical improvements and significantly outperforms the macro-learning system in [7].

In the rest of the paper, we review preliminaries, discuss interactions between macros, define observability of macros, describe the framework and our implementation, and present our experimental results and conclusions.

2 PRELIMINARIES

In the context of planning, *generalisation* is an operation that replaces *problem objects* by variables having identical names, but leaving the *domain constants* unchanged. Domain constants play designated specific roles in the domain dynamics. For instance, timber in the Settlers domain is specifically used in the definition of action build-sawmill. Further, a *generalised action* (pick ?b1 left in)) is obtained from a *grounded action* (pick b1 left in), where b1 is a problem object, ?b1 is a variable, and left and in are domain constants. Plans, action sequences, preconditions, effects, and parameters can all similarly be generalised from their grounded versions.

In this work, we consider only classical, sequential, and typed domains. Assume $\mathrm{P}(A) = \bigcup\{p\}$, $\mathrm{E}(A) = \bigcup\{p\}$, $\mathrm{V}(A) = \bigcup\{v\}$, and $\mathrm{C}(A) = \bigcup\{c\}$ respectively denote the precondition, effect, variable parameters (e.g. ?b1), and constant parameters (e.g. left and in) of a generalised action A. Here, v is a variable, c is a constant, and p is a literal (e.g. (pick ?b1 left in)) involving parameters from $\mathrm{V}(A) \cup \mathrm{C}(A)$. Further, positive and negative literals in $\mathrm{E}(A)$ respectively denote add and delete effects. Note that action header of a generalised action contains both variables and constants as parameters. Besides, constant parameters include the constants that are directly used in the preconditions and effects of the actions.

A *macro* $M = \langle A_1, A_2, \cdots, A_L \rangle$ is a totally-ordered sequence of L generalised actions. A macro M can also be represented by a resultant action if the action-sequence is *compiled* by using a well-known method called regression-based action composition. If $M = \langle A, B \rangle$ is a macro then $\mathrm{P}(M) = \mathrm{P}(A) \cup \bigcup_{p \in \mathrm{P}(B) \wedge p \notin \mathrm{E}(A)} \{p\}$, $\mathrm{E}(M) = \bigcup_{p \in \mathrm{E}(B) \wedge p \notin \mathrm{P}(A)} \{p\} \cup \bigcup_{p \in \mathrm{E}(A) \wedge \neg p \notin \mathrm{E}(B)} \{p\}$, $\mathrm{V}(M) = \mathrm{V}(A) \cup \mathrm{V}(B)$, and $\mathrm{C}(M) = \mathrm{C}(A) \cup \mathrm{C}(B)$. The resultant type of a variable parameter $v \in \mathrm{V}(M)$ is the more specific type (i.e. subtype) that v has in Action A or in Action B.

Notice that if $p \in \mathrm{E}(B)$ and $\neg p \in \mathrm{E}(A)$, we choose $p \in \mathrm{E}(M)$ over $\{\neg p, p\} \subseteq \mathrm{E}(M)$ although PDDL allows the latter. In PDDL, an action can have both $\neg p$ and p in its effect and the delete effects are applied before the add effects. This semantic is problematic for action composition and we do not adopt this. The reason is concatenating a sequence of more than two actions with p and $\neg p$ appearing alternately a number of times in them might result in an incorrect final effect. We therefore choose to consider only the final effects of such action sequences. This means we adopt a restricted action model where no action has both p and $\neg p$ at the same time in its effect.

A macro $M = \langle A, B \rangle$ is *invalid* if $\exists_p[(p \in \mathrm{P}(B) \wedge \neg p \in \mathrm{E}(A)) \vee (p \in \mathrm{E}(A) \wedge p \in \mathrm{E}(B))] \vee \exists_{p \neq q}[((p \in \mathrm{P}(M) \wedge q \in \mathrm{P}(M)) \vee (p \in \mathrm{E}(M) \wedge q \in \mathrm{E}(M))) \wedge \mathrm{mutex}(p, q)]$, otherwise M is *valid*. Here, $mutex(p, q)$ means p and q are *always mutually exclusive* in the domain dynamics. Notice that we define M to be invalid if $p \in \mathrm{E}(A) \wedge p \in \mathrm{E}(B)$ i.e. when the latter action achieves p while it is already achieved by the former. We explain below why in this case we do not consider p to be the resultant effect which could lead M to be considered as a valid macro.

A resource already acquired cannot be acquired again before releasing it, but a plate cleaned by a washer could be cleaned again even before making it dirty. Assume a literal p appears in both the former and the latter action. If $p =$ (acquired ?resource), this would mean acquiring the resource again by the latter action – which would be invalid. However, if $p =$ (clean ?plate) then the resultant effect would be p; here, the latter effect is redundant. Without a detailed domain analysis, it is not possible to know what kind of situation p represents. We assume that p is always of the first kind and we thus put more emphasis on sound reasoning rather than on flexible modelling.

Nevertheless, invalid macros could not always be detected in their generalised form. Two generalised literals $p(x)$ and $\neg p(y)$ could produce opposite grounded literals if both x and y, during planning, are grounded with the same problem object. Thus, a grounded macro containing them both could be invalid (according to the model we have adopted). Therefore, macros are replaced by the constituent actions in the plan and then the resulting plans are validated.

Parameter unification is an operator that replaces variables by type-compatible variables or domain constants. A specific type is *type-compatible* with any of its generic type in the type-hierarchy. Macros M and M' are the *same* macro, denoted by $M = M'$, if, after parameter unifications, their action sequences become identical. This requires variable parameters of one macro to be replaced by respective variable parameters of the other macro, and then vice versa. In this work, we do not consider macros to be the same when they have the same partially-ordered action sequence or have different action sequences but have equivalent preconditions and effects.

A *plan* is a totally-ordered grounded action sequence. Any subsequence of a generalised plan can thus form a macro. A macro M *occurs* in a generalised plan P (denoted by $M \vDash P$) if there exists an action subsequence M' in P such that $M = M'$. For convenience, we use M_p and M_s to denote the entire preceding and succeeding subsequences of M in P.

We define (grounded or generalised) *subgoals* in a generalised way. *Top-level subgoals* are the literals that directly appear in goal conditions of the problem instances of a domain. *Intermediate subgoals* are obtained by regressing top-level subgoals or other intermediate subgoals. Thus, if an effect of an action is a subgoal, the action's preconditions are also subgoals. A subgoal s *supports* another subgoal s' (denoted by $s \triangleright s'$) if s is an intermediate subgoal for s'; a macro M (or a generalised plan) having an effect s *achieves* s and so s' as well (denoted by $M \rightarrow s$ and $M \rightarrow s'$ respectively).

3 INTERACTIONS & OBSERVABILITY

We define three different types of interaction between macros. They help us understand aggregate behaviours of macro-sets. We then define unobserved macros w.r.t. a given set of plans.

Definition 1 (Conflicting Macros) *Given a generalised subgoal s and a generalised Plan $P \rightarrow s$, Macros $M \neq M'$ are* mutually conflicting *(denoted by $M \otimes M'$), if $M \vDash P \wedge M' \vDash P$, but due to overlapping actions, $M \vDash P \Longrightarrow [M' \nvDash M_\mathrm{p} \wedge M' \nvDash M_\mathrm{s}]$ and $M' \vDash P \Longrightarrow [M \nvDash M'_\mathrm{p} \wedge M \nvDash M'_\mathrm{s}]$. Selection of either macro thus hinders that of the other.*

Definition 2 (Competitive Macros) *Given a generalised subgoal s, Macros $M \neq M'$ are* mutually competitive *(denoted by $M \odot M'$), if $\exists_{P \rightarrow s, P' \rightarrow s}[M \vDash P, M' \vDash P']$; here, $P \neq P'$ are two generalised plans. Selection of one macro during planning would make the other redundant (w.r.t. achieving the subgoal).*

Definition 3 (Collaborative Macros) *Macros $M \neq M'$ are* mutually collaborative *(denoted by $M \oplus M'$), if $\exists_{M \rightarrow s \triangleright s^*, M' \rightarrow s' \triangleright s^*}[s \not\triangleright s' \wedge s' \not\triangleright s]$. Two collaborating macros, although individually address different subgoals, collectively help solve a higher-level subgoal or the whole problem.*

Consider the Settlers domain used in IPC 2004; a STRIPS encoding is described later. Macros PW = ⟨fell-timber, saw-wood⟩ and PS = ⟨build-quarry, break-stone⟩ produce respectively wood and stone locally while IS = ⟨break-stone, load stone, move, unload-stone⟩ produces stone somewhere else and imports. Another macro GH = ⟨produce-wood, import-stone, build-house⟩ gets a house built locally. Notice that PW $\oplus$ IS as PW $\rightarrow$ available-wood $\triangleright$ housing and IS $\rightarrow$ available-stone $\triangleright$ housing. Macro PS = ⟨build-quarry, break-stone⟩ produces stones locally. Notice that PS $\odot$ IS as both achieve available-stone. Assume $M_1 =$ ⟨break-stone, load stone⟩, $M_2 =$ ⟨move, unload-stone⟩, and $M_3 =$ ⟨load-stone, move, unload-stone⟩; here, M_1, M_2, M_3 all independently help achieve available-stone. However, $M_1 \otimes M_3$ as load-stone is an overlapping action, but $\neg(M_1 \otimes M_2)$ as their concatenation could achieve the subgoal available-stone.

Notice that the Settler problems have a number of various subgoals (some of which relate to *landmarks* i.e. subgoals that are to be achieved in all plans of a problem [8]). Achieving many such subgoals from other given subgoals requires sequences of more than one action and there exists more than one such sequence. Further, there are pairs of subgoals that do not support each other. For domains such as Settlers, a set of macros is therefore required, each macro in the set achieving a different subgoal. Moreover, the macros in the set should be *collaborative*, *non-competitive*, and *non-conflicting* with each other. Note that different interactions between any two given macros might not be mutually exclusive. This indicates the need of an empirical evaluation to determine the aggregate effect of a macro-set. Existing macro-learning systems such as [7] and [1] suggest only one or a given number of individually best performing macros without considering the interactions between the selected macros.

Definition 4 (Unobserved Macros) *Given a set of generalised plans S, a macro M is said to be* observed *from S if $\exists_{P \in S}[M \vDash P]$; otherwise, M is said to be* unobserved *from S.*

Consider the Gripper domain. Ball Bs (problem objects) are to be carried between two rooms I and O (domain constants) and the robot has two grippers L and R (domain constants). Consider the following plan of a gripper problem: $(\text{pick } B_0\ L\ O)$, $(\text{move } O\ I)$, $(\text{drop } B_0\ L\ I)$, $(\text{pick } B_1\ L\ I)$, $(\text{move } I\ O)$, $(\text{drop } B_1\ L\ O)$. Clearly, macros such as move-pick-move or move-drop-move cannot be observed from the plan although they achieve significant speedup in many problems. Given the above plan, these macros are neither generated by [7] nor by [1] (which uses a partial order lifting).

Of course the abovementioned two macros could be generated by concatenating domain actions with appropriate causal analysis; which is significantly different to learning macros from example plans. In this work, once we learn a macro such as pick-move-drop, we add random actions to it, or alter or delete its actions randomly and learn how the new macro performs. This mimics one kind of human learning – learn something, modify it slightly, experiment further, learn the new thing, and so on.

Nevertheless, generating necessary and sufficient training problems is very challenging and often requires detailed and specific domain/planner knowledge (such as two grippers and two rooms in the Gripper domain, the planner generates plans with the robot carrying one ball at a time, etc.). In this work, by generating unobserved macros, we attempt to address this issue to some extent.

4 THE FRAMEWORK

In the work [7], generalised macros are generated and evaluated for a given number of epochs; only the best-performing macros, however, survive through successive epochs. Initially macros in the current-working set are generated by *lifting* action-sequences from generalised plans of a small number of training problems (called *seeding* problems). Later, macros are also generated by applying the neighbourhood functions in Figure 1 on the macros in the current working-set. The neighbourhood functions generate only macros that occur in the given generalised plans. For example, the operators just *split* a macro into two, *delete* an action from either end of a macro, and *extend* a macro by the immediate preceding/succeeding action in the plans. During macro generation, many of the macros are discarded by a number of pruning rules that include limits on the parameter-count and the action-count.

Each letter in a sequence represents an action with its parameters

Plans	ABCDEFGHK | LMNPQ | Plans of seeding problems.
Macros	CDEFG | appears in 1st plan; an operand for operators.
Extend	BCDEFG | CDEFGH | B precedes; H succeeds CDEFG.
Shrink	CDEF | DEFG | Discard either end action of CDEFG.
Split	CDE | FG | CD | EFG | Randomly split CDEFG.
Lift	MNP | Lift randomly from a plan.

Figure 1. Restricted neighbourhood functions for macros.

For macro evaluation in [7], the action-sequence of a macro is compiled into a resultant action unifying only parameters that have identical names. Seeding problems have shared names for problem objects. The resultant action is then added to the domain to solve a number of other training problems (called *ranking* problems). The ranking problems are larger (just in solution times with no macro) than the seeding problems. The evaluation method then gives a numerical rating to the macro by using a fitness function $F(C, S, P) = C \times S \times P$ that involves three measures Cover (C), Score (S), and Point (P). *Cover* measures the portion of the training problems solved when the macro is used. *Score* measures a weighted mean time gain/loss over all the problems compared to when they are solved using no macro; any gain/loss for a larger problem gets more weight. *Point* measures the portion of the ranking problems solved with the macro taking less or equal time compared to when using no macro. A set of pruning rules are used during evaluation mainly to reduce the training time.

5 IMPLEMENTATION

The macro and macro-set spaces are exponential. Macros and macro-sets can have any numbers of actions and macros respectively. However, for practical reasons, certain maximum limits are imposed on the number of constituent actions in the macros, the number of unified parameters macros could have, and the number of constituent macros in the macro-sets. The macro space still remains huge and in the order of N^L, where N is the number of generalised actions and L is the number of actions in a macro. Similarly, the macro-set space remains in the order of M^C, where M is the number of macros in a pool and C is the number of macros in a macro-set. Any brute force or systematic but exhaustive search methods are therefore not suitable. In this work, we use a population-based stochastic local search algorithm described in Figure 2. As noted before, our algorithm uses the skeleton of the framework's algorithm.

Procedure Generalised-Macro-Learning
params: domain, planner, and T_S seeding and T_R ranking probs.
1. Solve the seeding problems using the domain and the planner. All these problems are solved within a certain time limit.
2. Generalise the plans replacing problem objects by variables, but keeping domain constants unchanged.
3. Call Individual-Macro-Learning (defined below).
4. Keep macros having fitness values greater than a threshold.
5. Call MacroSet-Learning (defined below).
6. Suggest the best macro-set as the output of the algorithm.

Procedure Individual-Macro-Learning
params: epoch-limit E_{I}, working-set size N_{I}, no-progress-limit P_{I}
1. W_{I}, the working-set for individual macros, is empty.
2. Repeat the following steps for E_{I} epochs.
   (a) Repeat the following steps for N_{I} times.
      i. Generate a new macro and add to the working-set W_{I}.
      ii. Exit if a previously-unexplored macro is not generated.
      iii. Evaluate the macro and assign a numerical rating.
   (b) Keep the best N_{I} (all in epoch 0) macros in W_{I}.
   (c) Exit if W_{I} remains the same for the last P_{I} epochs. Also, exit if a previously-unexplored macro is not generated.

Procedure MacroSet-Learning
params: epoch-limit E_{S}, working-set size N_{S}, no-progress-limit P_{S}
1. W_{S}, the working-set for macro-sets, is empty.
2. Repeat the following steps for E_{S} epochs.
   (a) Repeat the following steps for N_{S} times.
      i. Generate a new macro-set and add to the working-set W_{S}.
      ii. Exit if a previously-unexplored macro-set is not generated.
      iii. Evaluate the macro-set and assign a numerical rating.
   (b) Keep the best N_{S} (all in epoch 0) macro-sets in W_{S}.
   (c) Exit if W_{S} remains the same for the last P_{S} epochs. Also, exit if a previously-unexplored macro-set is not generated.

Figure 2. A comprehensive macro-learning algorithm.

The algorithm in Figure 2 has two phases. In the first phase, only individual macros are learnt using actions from generalised plans produced by solving given seeding problems. In the second phase, macro-sets are learnt using the macros (given as a *pool*) obtained in the first phase including only those that have fitness values more than a certain minimum level. During the learning process, macros and macro-sets are gradually generated and evaluated. In the generation method, new macros and macro-sets are generated using the neighbourhood functions on the macros and macro-sets in the current working-sets. In the evaluation method, given ranking problems are solved using the domains augmented by the compiled actions of the macros and macro-sets.

Representation. Macro-sets are represented by collections of macros (see Figure 3). Macros are represented by generalised action-sequences and resultant actions (see Figure 3). The action sequences are used in macro generation. First, new action sequences are generated by applying neighbourhood functions on current macros' constituent actions. Then, new sequences are compiled into resultant actions. Last, the resultant actions are added to the domains.

```
macro-set a-macro-set
    (a-macro)
    (another-macro)
    (yet-another-macro)

macro a-macro
    (move O I)
    (drop ?B L I)
    (move I O)

action a-macro
    parameters (?B - ball)
    precondition
        (robot-at-place O)
        (ball-at-gripper ?B L)
    effect
        (not (ball-at-gripper ?B L))
        (ball-at-place ?B I)
        (gripper-empty L)
```

Figure 3. Representation of macros and macro-sets.

Generation. For macro generation, we use the neighbourhood functions in Figure 1 that generate only observed macros. Also, we introduce a number of new functions (see Figure 4) that perform local search on the macro space and generate unobserved macros. For macro-set generation, we use the neighbourhood functions shown in Figure 5. For each macro or macro-set, the proposed functions ensure exploration of a large number of its neighbourhoods.

Each letter represents an action and its parameters

Plans	ABCDEFGHK | LMNPQ | RSTUVW | Generalised.
Macros	CDEFG (in 1st plan) | KQTV (a random operand)
Annex P	PCDEFG | CDEFGP | Add before or after CDEFG
Inject W	CWDEFG | CDWEFG | CDEWFG | CDEFWG
Delete	CEFG | CDFG | Delete an action in CDEFG
Alter	VDEFG | CDVFG | Replace an action by V
Concat	CDEFGKQTV | KQTVCDEFG | Concat. 2 macros.
Crossover	CDETV | KQFG | One's prefix & the other's suffix.
Assemble	DGMT | NVF | Accumulate actions randomly.

Figure 4. Extended neighbourhood functions for macros.

Note that any macro operand for any functions in Figures 1 and 4 always comes from the current working-set. An operand action is either lifted randomly from generalised plans of the seeding problems or collected randomly from the macros in the current working-set. Similarly, any macro-set operand for any functions in Figure 5 always comes from the current working-set; a macro operand randomly comes either from the given macro-pool or from the macro-sets in the current working-set. Operators are always selected according to user-defined probability distributions.

⋆ Each letter represents a macro; each string represents a set

Macro-Sets	NPQ | QRST | Operands for the operators
Add	MNPQ | MQRST | Add M to a macro-set
Drop	NP | RST | Drop Q from a macro-set
Change	NPW | QRWT | Replace a macro by W
Conjoin	NPQRST | Union of the two macro-sets
Disjoin	RT | QS | Split QRST into two macro-sets
Exchange	NST | PQR | Exchange macros between 2 sets
Gather	XYZ | Accumulate macros randomly
Top	UVW | A few top individual macros in the pool

Figure 5. Neighbourhood functions for macro-sets

The motivations behind selection of the neighbourhood functions are as follows: Good/bad solutions (macro or macro-set) normally remain in clusters. Discarding/adding/altering a good/bad component explores other solutions in the same cluster of a given solution. Combining good/bad components of two solutions finds a third good/bad solution. Constructing a solution from scratch ensures diversity of the exploration. Note, the neighbourhood function-sets are not minimal; effectiveness of any of them is not individually tested either.

Evaluation. The evaluation method produces an *augmented domain* for each macro or macro-set by adding the resultant action(s) into the original domain. It then solves the given ranking problems with the planner using both the original and the augmented domains under similar resource (e.g. time and memory) limits. The ranking problems do not take a long time to solve as they are to be solved for each macro or macro-set. Also, they are not solved so quickly as time gains may not be measured properly. The evaluation method then uses a fitness function F in Figure 6 to give a numerical rating to the macro or macro-set. F is a weighted sum of U and U', the fitness functions for solution time and plan-length respectively. Note that Weights W and W' are to be defined by the user. Each of U and U' involves three measures Cover (C), Score (S or S'), and Point (P or P'). Cover, Score, and Point are explained in Section The Framework. The intuition behind s_k (or s'_k) is clear from its values at certain points (*e.g.*, $s_k = 1$, $\frac{1}{2}$, and 0 for $\mu'_k = 0$, μ_k, and ∞ respectively). Moreover, its non-linear characteristic is suitable for a utility function. The weight w_k of gain/loss for the kth problem depends on its solution time while using the original domain. Among the three factors, Score plays the main role while Cover and Point mostly counterbalance any misleadingly high value. Notice that we give certain specific ratings to a macro or macro-set if $C = 0$ or if it causes (as explained in the Section Preliminaries) invalid plans to be produced. Invalid plans are detected by validating the plans both before and after expansion of the macros in the plans.

$F(U, U') = WU + W'U'$ given that $0 \leq W, W' \leq 1$ and $W + W' = 1$
$U(C, S, P) = C \times S \times P$ or $-\frac{1}{2}$ if $C = 0$ or -1 if invalid plans
$U'(C, S', P') = C \times S' \times P'$ or $-\frac{1}{2}$ if $C = 0$ or -1 if invalid plans

$S = \Sigma_{k=1}^{n} s_k w_k$ $\quad P = \Sigma_{k=1}^{n} p_k / n$ $\quad C = \Sigma_{k=1}^{n} c_k / n$
$S' = \Sigma_{k=1}^{n} s'_k w_k$ $\quad P' = \Sigma_{k=1}^{n} p'_k / n$ $\quad t = \Sigma_{k=1}^{n} t_k$
$s_k = t_k / (t_k + t'_k)$ $\quad s'_k = l_k / (l_k + l'_k)$ $\quad w_k = t_k / t$

$c_k = 1$ if prob-k is solved using the augmented domain or 0 otherwise
$p_k = 1$ if $t_k > t'_k$ (gain) or 0 if $t_k < t'_k$ (loss) or $\frac{1}{2}$ if $t_k = t'_k$ (tie)
$p'_k = 1$ if $l_k > l'_k$ (gain) or 0 if $l_k < l'_k$ (loss) or $\frac{1}{2}$ if $l_k = l'_k$ (tie)
Where,
n: #Probs to be solved using the original and augmented domains.
t_k, l_k: Plan-time, plan-length for prob-k while using the orig. domain.
t'_k, l'_k: Plan-time, plan-length for prob-k while using the aug. domain.

Figure 6. A fitness function for macro evaluation.

Pruning. Besides pruning invalid macros, we prune during macro generation other macros that have parameter-counts and action-counts more than the given limits. We also prune macros comprising actions that have no shared parameters among them (if arity is non-zero). To avoid repetitions, we check for duplicate macros. Macros that have different parameter names but their action sequences match when parameters are unified are also considered duplicate. During macro-set generation, we prune macro-sets that have macro-counts more than the given limit. Moreover, during evaluation, we prune macros/macro-sets, if they cause failure to solving a number of problems within given resource limits.

6 EXPERIMENTS

Experiments were run on several identical computers each having a Pentium 4 Dual Core 3GHz processor, 512KB cache memory, and 2GB physical memory. There were 5 seeding, 20 ranking, and 50 testing problems for each domain. The seeding and the ranking problems were randomly generated but were solvable in respectively 10 and 20 secs using 1GB memory; these were obtained by generating and solving problems until they were solved within the limits. To solve each testing problem (also randomly generated), the maximum time limit was 1800 secs while the memory limit was the same 1GB. For macro-set learning, the pool of macros was constructed using macros that have fitness levels more than 0.1. The limits on maximum action-count and parameter-count of macros were 8 and 6 respectively while that on macro-count of a macro-set was 6. Note, the search spaces still remained too huge to be explored exhaustively.

For both macro and macro-set learning, the maximum number of training epochs was 50 and the working-set size was twice the number of actions in the domain. Each neighbourhood function was selected with equal probability. The maximum number of attempts al-

lowed for each new macro or macro-set generation was 999,999; this was because, due to pruning rules, each attempt did not produce a macro or macro-set that requires evaluation. A macro or macro-set was pruned out during evaluation if more than 50% problems were not solved. The learning process was terminated when at least 1 replacement was not made in 10 epochs in a row.

Note, most parameter values were hand-crafted and were chosen intuitively; this was because the emphasis was not on improving learning efficiency rather on how the macros/macro-sets perform. Also note, IPC6 Learning Track allowed two weeks for training; the concern was how planning was improved when the learnt knowledge was exploited, no matter how long the training process had been run.

6.1 Results for Domains

We present results for the well-known state-of-the-art planner FF on benchmark domains such as Settlers[3] (24 actions), Rovers[4] (11 actions), Depots (5 actions), ReliefWork (12 actions), GoldMiner (7 actions), Gripper (3 actions), and Ferry (3 actions). FF is a heuristic-based forward search planner. Settlers, Rovers, and Depots are well-known IPC domains. ReliefWork is a domain written by us; it involves activities related to relief work scenarios in flood affected areas; the domain has actions for pick-up, patrol, and ambulance boats to provide food, shelter, and medical service to the victims. GoldMiner is used in IPC 2008 Learning Track. Gripper is used for a detailed study presented later. These domains are selected because they have more than one higher level generalised subgoal. Ferry is selected as a counterexample; it has only one top level subgoal (at ?car ?port) and subgoals are achieved by subsequences of only sail-embark-sail-debark-sail. Macro-sets and unobserved macros are not likely to perform well in Ferry. Note that these domains do not have actions with large parameter counts as macros comprising such actions will have very large numbers of grounded versions. We use a set of *testing problems* to demonstrate performance of the macros/macro-sets learnt. Most testing problems require 10 secs or more to be solved using no macro, and include instances that are not solved within given resource limits.

Figures 7, 8, and 10 report the experimental results. In the figures, N denotes the *original* domain that has *no macro*; O denotes the augmented domain for the *observed* macro learnt by [7]; E and C denote the augmented domains for the *efficient* and *combined* macro-set learnt by our method with ($W = 1.0, W' = 0$) and ($W = W' = 0.5$) respectively. For convenience, the learning processes related to the macros or macro-sets are also denoted by the same symbols. Nevertheless, E improves only solution speed while C strives to improve both solution-speed and plan-length (after macro expansion) of the plans produced with macros.

Figure 7 shows (left) the numbers of problems (out of 50) solved by domains N, O, E, and C. It also presents (right) macros' effect on plan lengths; note, in the plans produced, macros are substituted by respective action sequences. The plan quality shown in the figure is calculated by $100\% \times (L_N - L_M)/L_N$, where L_N and L_M respectively denote the plan lengths for N and M (M = O, E, or C). Figure 8 shows (left) the efficiency gains with macros or macro-sets learnt. We compare solution speeds achieved by E and C with that achieved by O; however, we also show comparison of

[3] We obtain a STRIPS encoding of the numerical version of IPC 2004 in the following ways: i) Discard metric functions such as labour, resource-use, and pollution; ii) Replace the other functions with corresponding predicates; iii) Convert increase and (assign to n) ($n > 0$) into add effects, and decrease and (assign to 0) into delete effects; and iv) Replace resource availability preconditions (e.g. $\geq$) with corresponding predicates.

[4] To avoid the opposite literals in the communicate actions, we introduce two actions for acquire and release channels.

Domain-	#Probs Solved				%PlanQualityGain		
	N	O	E	C	O	E	C
Settlers	24	34	46	40	-13	-36	-21
Rovers	36	42	50	50	-4	+7	+10
Depot	30	35	50	50	-4	-9	+4
ReliefWk	42	45	50	50	0	-4	+4
GoldMiner	36	44	50	50	-340	-29	0
Gripper	25	33	50	50	-38	-58	0
Ferry	50	50	45	43	-30	-50	-10

Figure 7. Performance in numbers of problems solved (left) and improvement in plan-quality with respect to Ns (right).

O and N. The efficiency gain shown in the figure is calculated by $100\% \times (L_O - L_M)/L_O$, where L_O and L_M respectively denote the plan lengths for O and M (M = N, E, or C). Nevertheless, Figure 8 also shows (right) the training times required.

Planner	%EfficiencyGain			Training CPU-Hr		
	N	E	C	O	E	C
Settlers	-500	97	69	41	69	89
Rovers	-337	91	88	101	120	137
Depot	-19	45	21	10	13	21
ReliefWk	-334	74	35	81	95	171
GoldMiner	-600	50	33	3	10	12
Gripper	-669	85	69	13	24	25
Ferry	-476	-28	-9	20	21	21

Figure 8. Performance gains in plan time i.e. efficiency with respect to Os (left) and training times (right).

To give further details, Figure 10 presents (left two columns) plan-time graphs for Settlers and Rovers. The figure also presents (the right most column) plan-length graph for GoldMiner. The results obtained for other domains are comparable with these results.

From the results (Ferry excluded), we find that both E and C solve more problems than O and achieve a magnitude faster speed than that achieved by O. However, C improves plan quality over O (even N) and E, but causes a significant loss in solution speed compared to E. We conclude both E and C significantly outperform O, spending training efforts roughly 1.5 to 2 times of that spent by O.

Domain-	#Probs Solved				%EfficiencyGain			
	E_I	O_S	E_T	O_T	E_I	O_S	E_T	O_T
Settlers	34	36	42	40	12	23	11	9
Rovers	42	44	41	41	33	66	48	43
Depot	33	38	31	25	10	40	-5	- 10
ReliefWk	45	46	45	45	2	9	4	4
GoldMiner	44	45	45	45	5	10	11	11
Gripper	33	39	35	35	15	33	36	30
Ferry	50	50	34	31	-12	-10	-18	-22

Figure 9. Performance in numbers of problems solved (left) and improvement in solution speed with respect to Os (right).

To study the effect of unobserved macros and macro-sets separately, we show in Figure 9 the best performances achieved with respective options turned on and off. In the figure, O_S denotes the macro-set learnt only from observed macros; E_I denotes the macro learnt when both observed and unobserved macros are explored. We found that O_S performs better than E_I which performs slightly better than O; E significantly outperforms O_S. This means learning unobserved-macros is not very beneficial; however, including them during macro-set exploration phase leads to learning macro-sets that achieve significant performance improvements over when they are not included. Overall, macro-set learning is found to be beneficial with or without unobserved macros.

We also compare macro-set learning with just using a number of top performing macros. In Figure 9, O_T denotes the macro-set comprising top k observed macros such that adding the $(k+1)$th observed macro would make the performance worse. E_T is like O_T but is learnt from both observed and unobserved macros. Both E_T and O_T perform slightly worse than O_S and significantly worse than E.

Figure 11 reports average sizes of macros (left) and macro-sets (middle); we can compare them with the limits set in the experiments.

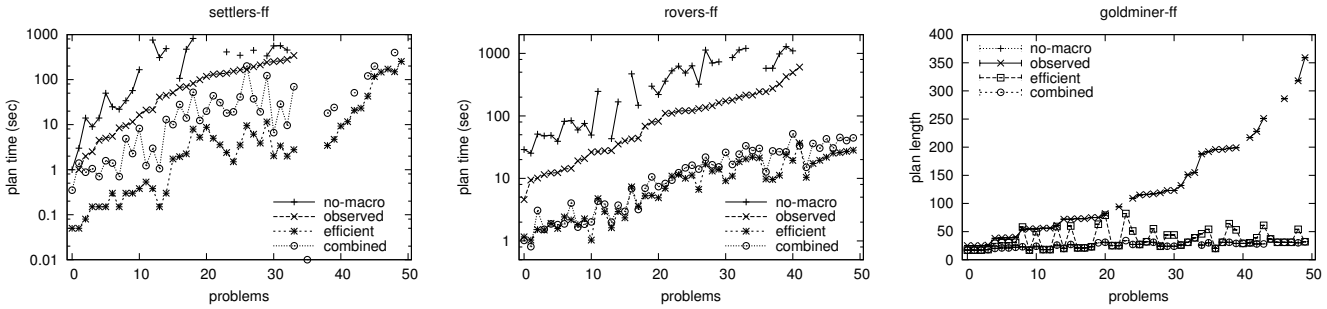

Figure 10. Sample performance: left 2 columns for plan-time, right column for plan-length; missing points: not solved

The averages are computed for macros in the last working-sets of the macro-set learning processes. Figure 11 also shows the (%) macros that are unobserved from plans of the seeding problems.

Domain	MacroSize			MSetSize		Unobs.	
	O	E	C	E	C	E	C
Settlers	5.0	5.6	5.8	5.8	5.6	75	60
Rovers	7.2	4.6	5.3	5.8	3.8	81	93
Depot	5.8	3.3	3.5	4.1	3.9	72	60
ReliefWk	5.4	4.8	3.0	5.0	4.4	90	97
Gripper	2.0	3.1	2.3	4.2	3.3	100	50
GoldMiner	5.6	5.4	5.0	4.0	3.6	79	36

Figure 11. Average sizes of macros (left), macro-sets (middle), and (%) Unobserved macros (right). Only macros in the last working-sets are used.

6.2 Analysis for Gripper

To be able to better explain our results, we choose a simple domain such as Gripper. The locations (I and O) and the grippers (L and R) are modelled as domain constants so that (at ?B I) and (at ?B O) are two distinct top-level generalised subgoals. Similarly, (move I O), (move O I) (pick ?B L I), (pick ?B L O), (pick ?B R I), (pick ?B R O), (drop ?B L I), (drop ?B L O), (drop ?B R I), and (drop ?B R O) are all distinct generalised actions. Note, the selection preference of a particular gripper to the other, when both are available, depends completely on the planner's internal processing order; any performance improvement due to this has no further explanation.

Macro P	(move O I) (drop ?B1 R I) (move I O)
Macro Q	(move I O) (pick ?B0 L O) (move O I)
Macro S	(move I O) (drop ?B2 R O) (move O I)
Macro T	(pick ?B4 L I) (move I O) (drop ?B4 L O) (move O I)

XYZ: Macro-Set {X,Y,Z}; X,Y,Z ∈ {P,Q,S,T}; No-Macro: N									
#	N	PQT	PT	QT	PQ	T	P	Q	TS
1	72	2.22	2.08	4.01	47	11.7	37	39	30
2	119	3.58	3.37	6.41	82	20.3	67	74	51
3	147	4.82	4.52	8.51	103	26.3	86	100	66
4	223	5.49	5.10	10.4	149	31.9	120	139	94
5	287	7.74	7.21	14.7	197	43.1	156	176	122
6	344	9.13	8.50	17.1	231	51.7	185	211	142

Figure 12. Performances of Gripper-FF macros and macro-sets based on solution times (in seconds) only. The problem instances have 300, 375, 450, 551, 650, and 751 balls respectively. Although performances are shown for only 6 problems, solution times exhibit the same trends for any numbers of problem instances with any numbers of balls.

Figure 12 shows performances of the macros and macro-sets for Gripper. It shows (top) the two best performing (based on solution time only, $W = 1, W' = 0$ in the fitness function) individual macros (T and S) and macro-sets (PQ and PQT). Further, it shows (bottom) the solution times when different macros and macro-sets are used, and also, when no macro is used. We used 5 seeding problems having respectively (5,4), (1,2), (2,6), (2,0), and (3,3) balls at (I, O) locations initially; the robot is initially at O in all problem instances with both grippers empty. One could verify that macros P and T can not be observed from FF-generated and then generalised plans.

The learnt macro-set PT obtains the greatest speed. Consider the drop actions of P and T. $P \rightarrow$ (at ?B I) and $T \rightarrow$ (at ?B O); thus $P \oplus T$. Further, $Q \rightarrow$ (at ?B I) (consider a following drop). Therefore, $Q \oplus T$. P and Q occur in two different plans that involve two different grippers; so $P \odot Q$. Macro-sets PQT (due to Q's overhead) and QT (due to P being better than Q individually) perform slightly worse than PT. Between PQT and QT, the former performs better as P individually performs better than Q and dominates in selections made by FF internally and thus reduces Q's overhead. Note, P and Q are both plateau-escaping macros (see [2]). The plateau is encountered when one ball is picked and a choice is to be made between (picking a second ball by the other gripper and moving) or (moving just with the ball already picked and dropping the ball). FF selects the latter option (relates to P). Macros T and S are the two best-performing individual macros learnt. However, $T \odot S$ as $T \rightarrow$ (at ?B O) and also $S \rightarrow$ (at ?B O); this is reflected in the performance of TS which is worse than that of T. T is the best performing individual macro as it achieves the subgoal from the associated initial state without requiring any other actions or macros.

7 CONCLUSION

We presented a macro-learning system that addresses three different issues of macro learning for planning. Our macro-learning system implicitly learns macro-sets based on their aggregate performances, explores unobserved macros besides the observable ones, and improve or reduce degradation of plan length when macros are used during planning. Our system significantly outperforms a recent macro-learning system both in solution speed and plan length.

Acknowledgements. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

[1] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer, 'Macro-FF: Improving AI planning with automatically learned macro-operators', *JAIR*, **24**, 581–621, (2005).

[2] Andrew Coles and Amanda Smith, 'MARVIN: A heuristic search planner with online macro-action learning', *Journal of Artificial Intelligence Research*, **28**, 119–156, (2007).

[3] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[4] Glenn A. Iba, 'A heuristic approach to the discovery of macro-operators', *Machine Learning*, **3**, 285–317, (1989).

[5] Richard E. Korf, 'Macro-operators: A weak method for learning', *Artificial Intelligence*, **26**, 35–77, (1985).

[6] Steven Minton, 'Selectively generalising plans for problem-solving', in *Proceedings of the IJCAI*, (1985).

[7] M A Hakim Newton, John Levine, Maria Fox, and Derek Long, 'Learning macro-actions for arbitrary planners and domains', in *Proceedings of the ICAPS*, (2007).

[8] J Porteous, L Sebastia, and J Hoffmann, 'On the extraction, ordering, and usage of landmarks in planning', in *Proceedings of the ECP*, (2001).