Learning Macro-Actions to Improve the Relaxed Planning Graph Heuristic

Dancheng Gao, Andrew Coles, Amanda Coles
Department of Informatics
Faculty of Natural, Mathematical and Engineering Sciences
King's College London
30 Aldwych
London WC2B 4BG
dancheng.gao@kcl.ac.uk, andrew.coles@kcl.ac.uk, amanda.coles@kcl.ac.uk

Abstract

Research has shown that using macro-actions in planning often helps to improve the performance of planners. However, their use comes at the cost of an increased branching factor during search, which may affect the planners' effectiveness adversely. Since macro-actions also influence heuristic computation, removing them from search while keeping them in the heuristic may still be able to help improve the planner's performance. This paper aims to assess the performance of planning under the Relaxed Planning Graph heuristic when macro-actions are excluded from the search process. Useful macro-actions for benchmark domains were learned using two different approaches, one systematic and one inspired by ParamILS. Empirical evaluation shows improvement in a number of domains.

1 Introduction

Current AI planners have been proved successful in solving a range of problems. However, harder problems can still present a challenge to scalability, and macro-actions are one option to help with this (Fikes, Hart, and Nilsson 1972). A macro-action is a meta-action built from a sequence of action steps that, when applied, generates a successor state equivalent to applying each action in turn (Korf 1985). By performing search with a domain augmented with these macro-actions, the result is a search space that is broader – there is an increased branching factor – but, all being well, needs to be explored less deeply. This trade-off between breadth and depth underpins the importance of choosing good macro-actions.

In addition to their historic intention of modifying the search space directly (Dawson and Siklóssy 1977), extending a domain with macro-actions influences the heuristic, and may modify planner performance in this way (Coles and Smith 2007). But, a process for learning macro-actions for use in a given domain, with a given planner, may well then be conflating these two effects: the benefit or drawback of a given macro-action may be due to its effect on the search space, or the heuristic, or both.

In this paper, we explore the effect of macro-actions on the planning process when they are included in the domain, but are *excluded from search*. This removes any effect on branching factor, with the focus being on macro-actions that support better heuristic guidance. Using a macro-action generator that is able to generate and evaluate macro-actions for arbitrary domains, we describe two learning methods – one more systematic, one using a local-search approach inspired by ParamILS (Hutter et al. 2009) – and use these to learn useful macro-actions with the Fast Downward planner (Helmert 2006). Experiment results show that the learnt macro-actions can help to improve the performance in multiple benchmark domains.

2 Background

In this paper we consider deterministic classical PDDL (Fox and Long 2003) planning, with our focus being on quickly finding a plan that achieves the desired goals. A popular approach to solving planning problems is to perform forward search over a state-space defined by the actions in a planning domain, and guided by a heuristic. The labels along a path from the initial state to a state in which the goals are satisfied is a plan.

2.1 Heuristics

In planning problems, a heuristic provides an estimate of the distance from the current state to the goal state (McDermott 1996). As calculating the real heuristic value is a NP-hard problem (Bonet and Geffner 2001), a relaxed version of the original planning problem is commonly used to compute a heuristic value. A number of heuristics have been proposed; we use the Relaxed Planning Graph (RPG) heuristic in this work (Hoffmann 2001), which is based on a delete-relaxation (all negative effects of the actions are ignored). The RPG heuristic can find a (non-optimal) relaxed plan in polynomial time, and the number of actions in this relaxed plan serves as a domain-independent heuristic value.

One planner that can make use of the RPG heuristic is Fast Downward (Helmert 2006). Fast Downward translates PDDL domains into the SAS$^+$ formalism (Bäckström and Nebel 1995) before the actual search process (Helmert 2009). SAS$^+$ allows planning tasks to be expressed using multi-valued state variables, facilitating analyses (and a compact state representation) that improve planner performance.

2.2 Macro-actions

As a domain-independent heuristic will work more or less well in different domains, one option to improve planner

performance in a specific domain is to learn macro-actions. A macro-action in AI planning is a meta-action built from a series of actions. Applying a macro-action to a state is equivalent to applying the sequence of actions that build up the macro-action. Macro-actions typically influence planner performance in two ways:

- By modifying the search space: states have additional successors corresponding to the application of macro-actions. Thus, by learning macro-actions that are based on action sequences commonly seen in solution plans, one can aim to reduce search effort by providing shortcuts through the search space.
- By modifying the heuristic: with a heuristic such as the RPG heuristic that is based on the given domain, the addition of macro-actions to a domain will influence the heuristic.

A caveat for the use of macro-actions is that more actions need to be considered when finding the successors of each state. This means that the branching factor in each iteration of the search process is increased and more states need to be evaluated. Too many macro-actions can result in a drastic increase in the search space breadth, and a severe decrease in the planner's performance.

2.3 Generating macro-actions for arbitrary planners and domains

Several macro-actions generation techniques have been explored in the past, including off-line learning such as Macro-FF (Botea et al. 2005; Botea, Müller, and Schaeffer 2004), and online generation such as Marvin (Coles and Smith 2007). Wizard is another approach, learning macro-actions using a genetic framework. Wizard generates macro-actions by using the original domains to solve small problems, and lifting action sequences from the plans (Newton and Levine 2007b; Newton et al. 2007). Since macro-actions not used in any plans can still help to improve the planner's performance, Wizard is also designed to learn unobserved macro-actions. This is done by evolving the observed macro-actions (Newton and Levine 2007a). More recent approaches include MEvo (Vallati, Chrpa, and Serina 2020), which dynamically selects effective macro-actions from a given pool, and a new technique to generate Critical Section macros (Chrpa and Vallati 2021).

2.4 ParamILS

As complex algorithms have many parameters, with multiple possible values, it is usually too time-consuming to exhaustively look for the best combination of parameters. ParamILS (Hutter et al. 2009) approaches this challenge by using local search to explore the parameter space of an algorithm to identify parameter settings that yield good performance. To summarise the terminology and parameters of ParamILS:

- θ: a single configuration (combination of parameters).
- Incumbent: the best performing configuration.
- $\kappa_{\max}$: maximal allowed run time of a configuration. The cutoff run time of a configuration must be equal to or lower than this value.
- objective(θ, N, optional parameter bound): computes the mean run time of a configuration over N number of problem instances. The optional parameter bound is the maximum allowed objective value. When this value is not specified, a large integer value is assumed to be the bound.
- dominates(θ_1, θ_2): configuration θ_1 dominates θ_2 when the N value of θ_1 is at least as large as that of θ_2, and when the objective value of θ_1 is smaller or equal to that of θ_2.
- Iterative first improvement(θ): a process to recursively find the best configuration θ_{ils} in the neighbourhood of θ.
- Neighbourhood(θ): the neighbourhood of a configuration θ is defined by the set of all configurations that differ from θ in one parameter.
- s: the number of times to choose a configuration during perturbation.
- p_{restart}: the probability to restart the process.
- r: the number of configuration considered before the iterative first improvement process.

ParamILS makes use of randomness and an adaptive capping mechanism in focused local search, and tries to find the best combination in a short period of time. Configurations are compared using objective values. The N number increases when two configurations have the same N value, and when one configuration is dominating the other. An initial incumbent is chosen from a random r number of configurations. Iterative first improvement is then performed on the chosen incumbent, θ_{ils}. The perturbation process then starts by randomly choosing a configuration from the neighbourhood of θ_{ils} s number of times, and performing iterative first improvement on the chosen configuration. If the resulting configuration θ of iterative first improvement is better than θ_{ils}, θ_{ils} is replaced by θ. θ_{ils} can be replaced by a random configuration with the probability p_{restart}, until the termination criterion is met. Empirical results have shown that as compared to the default input configurations, ParamILS is able to find significantly better configurations for various SAT algorithms and CPLEX benchmark sets.

3 Generating Macro-actions

Macro-actions are not originally included in the benchmark domains. Therefore, in order to assess the effect of macro-actions, the domains have to be processed to generate macro-actions, and these are embedded into an extended domain definition that we refer to as an *augmented domain*. The macro-action generation process we used is briefly described below:

1. Merge a sequence of two actions together, with the preconditions being the weakest preconditions of the sequence (those that must be true for it to succeed) and the effects being the strongest effects (changes to the

states that necessarily occur once the sequence has been applied). By way of example, the Gripper domain comprises actions to 'pick' up and 'drop' balls, and to 'move' a robot from one room to another. If the sequence to 'pick' then 'move' is merged, and the parameters of 'pick' are renamed to ensure they are distinct to those of 'move', we obtain the following:

```
(:action macro-pick-and-move
    :parameters (?b_0 - ball ?g_0 - gripper
                 ?p_0 ?fp ?tp - place)
    :precondition (and
                (ball-at-place ?b_0 ?p_0)
                (robot-at-place ?p_0)
                (gripper-empty ?g_0)
                (robot-at-place ?fp)
                (not (= ?fp ?tp)))
    :effect (and
                (not (ball-at-place ?b_0 ?p_0))
                (not (gripper-empty ?g_0))
                (ball-at-gripper ?b_0 ?g_0)
                (not (robot-at-place ?fp))
                (robot-at-place ?tp)))
```

2. Create several instances of this macro action, considering the different combinations of parameter bindings. Without referring to a solution plan and/or problem file, it is impossible to know whether same-type parameters should be assigned the same or different objects at planning time. Therefore, all possibilities should be considered. For each pair of parameters ?a and ?b with the same type, create a pair of predicates `(= ?a ?b)` and `(not (= ?a ?b))`. In the 'macro-pick-and-move' example, the following pairs will be created:

```
pair 1: (= ?p_0 ?fp) , (not (= ?p_0 ?fp))
pair 2: (= ?fp ?tp) ,  (not (= ?fp ?tp))
pair 3: (= ?p_0 ?tp) , (not (= ?p_0 ?tp))
```

   An instance of the macro action then has one predicate from each pair, so in this example, there are 8 possible combinations. For instance:

```
(not(= ?p_0 ?fp)) (not(= ?fp ?tp)) (= ?p_0 ?tp)
```

   For each combination, take the base macro action, and add the predicates needed to define the combination. The result here, thus, is 8 macro-actions, differing only in terms of the parameter relationships between the underlying actions.

3. Refine the created macro instances, again retaining only weakest preconditions and strongest effects – these may bave changed once the parameter bindings have been set. As a simplification, if two parameters are set to be equal in the precondition, one of the parameters will be replaced by the other throughout the action, and the equality precondition removed. An example of the refined macro instance generated after this step, using the previous parameter constraint combination, is shown below:

```
(:action macro-pick-and-move_7
    :parameters (?b_0 - ball ?g_0 - gripper
                 ?p_0 ?fp - place)
    :precondition (and
                (ball-at-place ?b_0 ?p_0)
                (robot-at-place ?p_0)
                (gripper-empty ?g_0)
                (robot-at-place ?fp)
                (not (= ?fp ?p_0)))
    :effect (and (not (ball-at-place ?b_0 ?p_0))
                 (not (gripper-empty ?g_0))
                 (ball-at-gripper ?b_0 ?g_0)
                 (not (robot-at-place ?fp))
                 (robot-at-place ?p_0)))
```

4 Learning Useful Macro-actions Systematically

We begin by describing a way of learning macro actions to explore whether it is possible to use these to improve planner performance without being used in search. We do not refer to solution plans in our approach here – as our macro-actions are not to be used in search, it follows that good macro-actions do not necessarily resemble solution plans. Hence, we use a multi-round approach that in the first round performs a systematic evaluation of all macro-actions comprising two primitive actions; and run this for a number of iterations to expand the number of macro-actions in the domain further. The learnt macro-actions are then evaluated using larger problem files to assess their performance.

4.1 Learning Process

Systematic learning consists of different rounds. The initial round only considers macro-actions that are formed by using two actions from the original domain. Each subsequent round will make use of the result of the previous round, i.e. the best performing domain (including its macro-actions) of one round is used as the basis for macro-actions in the next. Note that since the best domain after each round is fixed to be used as a basis for subsequent rounds, this systematic learning approach is not an exhaustive one. In this paper, the systematic learning process is run for three rounds for each benchmark domain.

The best performing domain of each round is selected by comparing total planning times on three training problems. Fast Downward first translates the augmented domain into an SAS$^+$ output file. The SAS$^+$ output is checked to see if it contains any macro-actions. If no macro-actions are included, it means that the macro-actions in the current augmented domain were inconsistent and/or not useful, so the augmented domain will therefore be discarded. Each training problem is solved once for each augmented domain, with a time limit of 900 seconds. Once all training problems are solved for an augmented domain, its total planning time will be computed. After all problems have been solved for all augmented domains, the augmented domain with the lowest total planning time for all training problems will be selected as the best domain of this round.

The initial objective of this paper is to find macro-actions that are useful during heuristic computation. This kind of macro-actions are more likely to change the initial heuristic values of the planning process. Therefore, depending on the experimental setups, the initial heuristic value of a plan-

	Original	Baseline		No-initial-checking	5s
		Macro	Translator-only		
Depots	9.93	**14.97**	12.17	13.72	14.73
Driverlog	16.21	15.55	15.46	**18.74**	18.46
Rovers	17.17	12.25	16.89	16.88	**17.38**
Satellite	17.30	16.75	**18.59**	18.17	-
ZenoTravel	18.96	17.98	**19.44**	18.31	17.00

Table 1: IPC scores of the different systematic learning configurations for benchmark problems.

	Original	Baseline		No-initial-checking	5s
		Macro	Translator-only		
Depots	11.58	**22.35**	14.56	18.60	22.11
Driverlog	19.01	25.25	21.35	**29.79**	29.24
Rovers	19.56	15.10	20.20	20.48	**21.80**
Satellite	20.35	23.39	**26.71**	25.70	-
ZenoTravel	26.79	27.78	**30.58**	28.25	26.63

Table 2: IPC scores of the different systematic learning configurations for all evaluating problems (including both benchmark and more complex problems).

ning problem may be checked to determine whether an augmented domain is worth examining or not.

4.2 Experimental Setups

Fast Downward is the planner used for all experiments. The search method selected is lazy greedy best first search, and the heuristic method used is the RPG heuristic (the 'FF' heuristic). The only modification made to the planner is identifying macro-actions and excluding them from the search process.

The domain used for the experiments are IPC2002 (Long and Fox 2003) benchmark domains, namely Depots, Driverlog, Rovers, Satellite, and ZenoTravel. Three problem files that take less than 30 seconds to solve with the original domain model are randomly generated using problem generators, to be used as training problems. All experiments used a 2GHz Xeon E5-2660 CPU, with an 8GB memory limit.

Multiple sets of experiments are run using different learning configurations. Each set of experiments is run for three rounds for each domain. The best planning time of a round is the total planning time of the best performing domain of the round. The names and set-ups of these configurations are listed below:

1. **Original**: the original, unmodified domain.
2. **Baseline:Macro**: including learnt macro-actions, with the initial heuristic values being checked during the learning process: for a given problem instance, if the initial heuristic value of an augmented domain is the same as that of the original domain, the augmented domain is assumed to have the same planning time as the original domain. The problem files used for the learning process originally take 15-30 seconds to solve.
3. **No-initial-checking**: the initial heuristic values are not checked during the learning process. The problem files used for the learning process originally take 15-30 seconds to solve.
4. **5s**: the problem files used for the learning process originally take 5-10 seconds to solve. These problem files are randomly generated using the problem generators.

For each benchmark domain, we compare the best total planning time on the three training problems after each of the three rounds, and the best is selected as the learnt domain. Benchmark problems as well as more complex problems that originally take more than 70 seconds to solve are used for evaluating the learnt domains. The more complex problems are randomly generated using problem generators. As our focus is on reducing planning time, our overall evaluation metric is that of the 'agile' track in the 2014 IPC[1], where the score of a given planner on a given problem is 0 if it did not solve it, or:

$$1/(1 + log(T/T^\star))$$

...where $T^\star$ is the lowest planning time seen by any planner, and T is that of the current planner. Any times below 1 second get the same score.

During evaluation, it is observed that apart from changing the initial heuristic values of planning problems, macro-actions are also able to change the translated SAS$^+$ encoding. Therefore, to assess the effect of macro-actions on SAS$^+$ encoding, an additional **Baseline:Translator-only** configuration is used where macro-actions are removed from the SAS$^+$ outputs after translation. In this way, macro-actions are only allowed to change the SAS$^+$ encoding, and have no effect on the heuristic computation process.

4.3 Experiment Results

Table 1 shows the IPC scores of the benchmark competition problems for each of our evaluation configurations. Note the

[1] https://helios.hud.ac.uk/scommv/IPC-14/rules.html

	Original	Round 1	Round 2	Round 3
Depots	108.53	33.38	**1.45**	13.78
Driverlog	91.85	7.03	**4.08**	6.30
Rovers	98.49	24.11	0.12	**0.10**
Satellite	112.17	46.67	22.71	**20.01**
ZenoTravel	109.89	58.67	55.64	**27.59**

Table 3: Best planning time (seconds) of each learning round, when the initial heuristic values are checked during the learning process.

	Round 1	Round 2	Round 3
Depots	2.31/16.12	11.25/33.23	22.81/52.80
Driverlog	2.69/9.17	6.52/28.15	22.45/49.10
Rovers	15.75/82.42	31.11/191.62	67.96/272.28
Satellite	4.15/8.36	6.21/10.62	35.67/18.79
ZenoTravel	23.37/38.53	70.99/52.48	128.55/55.51

Table 4: Comparing the learning times (hours) of each round for Baseline/No-initial-checking configurations.

IPC scores of our augmented domains may be worse than the original domain. This is because a significant portion of the benchmark problems are small problems that take less than 1 seconds to solve. Including macro-actions for these problems will create an overhead that slows down the planning process, since the translator and heuristic need more time to consider macro-actions. To better illustrate performance on non-trivial problems, we thus randomly generated additional problems that originally took more than 70 seconds to solve. The IPC scores when more complex problems are included are shown in Table 2.

4.3.1 The effect of macro-actions on the performance of the planner In this set of experiments, the Baseline:Macro configuration is used. The planning times of the original domain and those of the augmented domains with macro-actions are compared. The purpose of this set of experiments is to see whether the learnt macro-actions are able to improve the performance of the planner.

Table 3 shows the best planning time of each learning round on the training problems, showing that the performance of macro-actions on these problems is significantly improved compared to the original domain. In Table 1 and 2, by comparing the Original column and the Macro column, it can be seen that the Depots domain has shown improvements in both the benchmark and more complex problems, while the Driverlog, Satellite, and ZenoTravel domains only have improvements in the more complex problems. The learnt macro-actions are not useful in the Rovers domain. Nonetheless, this serves as first confirmation that it is possible to find macro-actions that are useful outwith search.

4.3.2 The effect of macro-actions on the SAS$^+$ encoding As noted in section 4.2, macro-actions can change the SAS$^+$ translation of a planning problem. Since Fast Downward relies on SAS$^+$ encoding to solve planning problems, the change in SAS$^+$ output from the translator can result in different planner performance. Therefore, the purpose of this set of experiments is to examine whether the change in SAS$^+$ encoding brought about by the learnt macro-actions can help to improve the performance of the planner. The same macro-actions learnt in section 4.3.1 are used again for evaluation, but any ground macro-actions are removed from the SAS$^+$ output from the translator prior to running the planner itself. The evaluation results can be found in the Translator-only column in Table 1 and 2.

By looking at the Original column and the Translator-only column in Table 1 and 2, the planning times of the original SAS$^+$ encoding and those of the augmented SAS$^+$ encoding without macro-actions are compared. It can be seen that the Depots, Satellite, and ZenoTravel domains have improvements in both the benchmark and more complex problems, while the Driverlog and Rovers domains only have improvements in the more complex problems. Either way, even without the heuristic having access to macro-actions, the change of encoding alone can improve planner performance

Much as adding macro-actions to search can reduce planner performance, by increasing the branching factor, adding macro-actions for use by the heuristic can increase the overheads in computing heuristic values. To explore this trade-off, by comparing the Macro and Translator-only columns in Tables 1 and 2, it can be seen that for the Rovers, Satellite, and ZenoTravel domains, macro-actions are more effective in improving just the SAS$^+$ encoding, whereas for the other two domains the best performance is when these are available for use by the heuristic. Different domains have different performance due to the fact that mixed types of macro-actions are learnt during the learning process: some of them are more useful in the heuristic, while the others are more useful in improving the SAS$^+$ encoding.

4.3.3 Including macro-actions with the same initial heuristic value in the learning process In previous experiments, augmented domains with the same initial heuristic value for the same problem instance are assumed to have the same planning time. This is a short-cut with the aim to reduce learning time, but may exclude macro-actions – yielding different heuristic values in other states – that actually have better planning times. As a result, it is possible that some useful macro-actions are left out during the learning process in previous experiments. In this set of experiments, the No-initial-checking configuration is used – all macro-actions are evaluated in full, regardless of initial state heuristic values. This will, however, significantly increase the time taken to learn macro-actions.

The learning times for each round of the learning process for both Baseline and No-initial-checking configurations are listed in Table 4; as can be seen, the No-initial-checking configuration has higher learning times for most domains. The exceptions are Satellite and ZenoTravel, where the No-initial-checking configuration has a shorter learning time in the last or second last rounds. This is because different macro-actions are learnt in the previous rounds, which leads to different number of macro-actions being generated in the next round.

By comparing the Macro column and the No-initial-checking column in Table 1 and 2, it can be seen that the macro-actions learnt for the Driverlog, Rovers, Satellite, and ZenoTravel domains are better than those produced by the Baseline configuration for both benchmark and more complex problems. The learnt macro-actions for the Depots domain are worse than those learnt by the Baseline configuration, but the planner still has better performance than the original domain. The results show that the No-initial-checking can generally produce better macro-actions than the Baseline configuration, at the cost of significantly longer learning times. The only exception is the Depots domain. This is because many macro-actions have similar performance during the learning process. As a result, different macro-actions may be learnt in the first round, which leads to different macro-actions being generated in the subsequent rounds. Therefore, there is a chance that the No-initial-checking approach ultimately generates worse macro-actions than the Baseline approach.

4.3.4 Using smaller problems to learn macro-actions In order to reduce the time taken to learn macro-actions, smaller training problems are used. In this set of experiments, the 5s configuration is used, which means that the generated training problems originally take 5-10 seconds to solve. The experiments aim to find out whether using smaller training problem files has an impact on the performance of the planner.

By comparing the Macro column and the 5s column in Table 1 and 2, it can be seen that whilst smaller problems did not allow any useful macro-actions to be learnt in Satellite, the macro-actions produced by this configuration were better for Depots, Driverlog and Rovers, and slightly worse for the ZenoTravel domain.

5 Learning Useful Macro-actions Using Local Search

Not all learnt macro-actions from the systematic approach perform well during evaluation. Especially for the Rovers domain, the learnt macro-actions provide limited or no improvement over all configurations. This can be due to the fact that only three problem files are used for learning, which can lead to over-fitting issues. Therefore, including more problem files during the learning process may help to generate more useful macro-actions. However, our systematic learning process is time consuming, as many macro-actions have to be considered. To address this problem, this paper presents a second learning method inspired by ParamILS.

5.1 Learning Process

Some of the definitions from ParamILS that are modified during the learning process are listed below:

- θ: a single domain. Augmented domains with various number of macro-actions are the domains that need to be tested.
- Incumbent: the best performing domain.
- $\kappa_{\max}$: the maximal captime is set to 30 seconds.
- objective(θ, N, optional parameter bound): the objective value of a domain is calculated by either the mean or the median planning time across N training problems. Since there is at most 20 training problems, the N number is capped at 20.
- dominates(θ_1, θ_2): for θ_1 to dominate θ_2, the objective value of θ_1 has to be significantly better than that of θ_2. When the objective value is calculated by the mean planning time, θ_1 and θ_2 are significantly different if the difference between the sum of planning times of θ_1 and that of θ_2 across N problems is at least [d / (total number of training problems / N)]. When the objective value is calculated by the median planning time, θ_1 and θ_2 are significantly different if the difference between their objective values is at least 1.
- Neighbourhood(θ): The original domain is used to generate an initial set of macro-actions which only consist of two primitive actions. The neighbourhood of a domain consists of the following two types of domains:
  1. Remove one macro-action from the current domain, add one other macro-action from the initial set of macro-actions.
  2. Keep all macro-actions in the current domain, add one other macro-action from the initial set of macro-actions.

  Additionally, we set a limit m to be the maximum number of macro-actions allowed in a domain.
- Iterative first improvement(θ): since the neighbourhood sizes are too large for the augmented domains, the process is modified to only consider a randomly selected n number of neighbours, rather than iterating through the entire neighbourhood. (NB at least one neighbour of each of the two types identified above is kept.)
- s: the value is set to 3.
- p_{restart}: the value is set to 0.01.
- r: the value varies for different experiments.

The learning process terminates either when the current incumbent is performing significantly better than the original domain, or when domains have more than m number of macro-actions.

5.2 Experimental Setups

The same domains, planner, and evaluating problems used for systematic learning are used again in this learning approach. Multiple sets of experiments are run using different configurations. Since randomness is involved in this learning method, each set of experiments is run 10 times to improve the accuracy of the experiments. The best learning outcome out of 10 runs for each domain is selected for evaluation. An ablation study is to assess the impact of different parameters on the learning outcomes. The names and set-ups of these configurations are listed below:

1. Original: The original unmodified domain
2. Baseline: The r, d, m, and n values are set to 20, 10, 5, and 10 respectively. Objective values calculated using mean planning times. Twenty problem files that can be solved

	Original	Baseline	r		d		m		n		3-problems	Longer-macros
			5	10	5	20	1	3	5	20		
Depots	9.93	10.58	10.32	9.90	9.85	9.91	10.23	9.96	9.91	9.90	**14.52**	9.91
Driverlog	16.21	16.18	16.25	16.16	16.17	16.79	15.98	16.13	16.15	16.14	**17.41**	16.14
Rovers	17.17	17.28	17.13	17.06	17.05	17.10	17.05	17.09	16.96	17.88	14.46	**17.93**
Satellite	17.30	17.89	16.46	17.86	17.77	17.81	17.87	18.11	17.73	**18.67**	17.98	18.52
ZenoTravel	**18.96**	17.23	18.68	18.03	18.68	17.89	18.05	18.41	18.52	17.45	17.18	18.07

Table 5: IPC scores of the different ParamILS-inspired learning configurations for benchmark problems, when objectives are calculated by mean.

	Original	Baseline	r		d		m		n		3-problems	Longer-macros
			5	10	5	20	1	3	5	20		
Depots	11.58	11.89	11.99	11.54	11.46	11.55	11.54	11.61	11.55	11.55	**22.77**	11.55
Driverlog	19.01	19.26	20.56	18.94	19.25	**25.47**	22.67	18.90	19.97	18.92	22.53	24.76
Rovers	19.56	24.09	19.60	20.90	24.55	20.14	24.55	18.75	20.47	**26.77**	15.78	24.13
Satellite	20.35	**28.67**	23.13	26.19	28.46	28.53	25.84	25.38	27.28	25.59	25.77	26.56
ZenoTravel	26.79	27.19	26.41	28.30	26.46	27.92	28.38	**29.07**	26.23	28.12	26.87	28.44

Table 6: IPC scores of the different ParamILS-inspired learning configurations for all evaluating problems (including both benchmark and more complex problems), when objectives are calculated by mean.

by the original domain under 30 seconds are randomly generated to be the training problems.

3. r-test: r values are set to 5 and 10 respectively.
4. d-test: d values are set to 5 and 20 respectively.
5. m-test: m values are set to 1 and 3 respectively.
6. n-test: n values are set to 5 and 20 respectively.
7. Median: Median instead of mean planning times are used as objective values.
8. 3-problems: Only 3 problem files are used for learning. The 3 problem files are the same as those used in systematic learning.
9. Longer-macros: For each domain θ that requires neighbourhood generation, use all the macro-actions in θ to generate a new set of longer macro-actions. The augmented set of macro-actions consists of the initial set of macro-actions, and all newly generated macro-actions. The neighbourhood of θ is modified to include the following two types of domains:

   (a) Remove one macro-action from the current domain, add one other macro-action from the augmented set of macro-actions.

   (b) Keep all macro-actions in the current domain, add one other macro-action from the augmented set of macro-actions.

5.3 Experimental Results

5.3.1 Learning macro-actions using this approach In this set of experiments, the Baseline configuration is used, and compared to Original. The aim of this set of experiments is to demonstrate that the new learning methods is able to produce useful macro-actions.

As seen from Table 5, the Baseline configuration is able to provide useful macro-actions for the benchmark problems for the Depots, Rovers, and Satellite domains. The effect on benchmark problems is not very obvious for the Driverlog domain, and the Zenotravel domain has worse performance. For the more complex problems, as shown in Table 6, all domains have better performance as compared to the original domain. This proves that the new learning method is able to produce useful macro-actions, especially for larger problems. This is consistent with the observation made in Section 4.3, where macro-actions only paid off on the more complex problems.

5.3.2 Sensitivity tests for r, d, m, and n The values of r, d, m, and n are varied to inspect their impact on the learning outcome. As the benchmark problems are too small to be used for meaningful evaluation here, our focus will be on the more complex problems.

By comparing the r column with the Baseline column in Table 6, it can be seen that the performance of the Rovers and Satellite domain improves as the r value increases. The r value does not have significant impact on the other domains.

By comparing the d column with the Baseline column in Table 6, it can be seen that the Driverlog and ZenoTravel domains have better performance as the d value increases. The Rovers domain has worse performance as the d value increases. The d value dose not have significant impact on the other domains.

By comparing the m column with the Baseline column in Table 6, it can be seen that only the Depots domain has slightly better performance as the m value increases – learning always terminated with fewer than 5 macro-actions.

By comparing the n column with the Baseline column in Table 6, it can be seen that the performance of the Rovers and ZenoTravel domain improves as the value of n increases. It does not have significant impact on the other domains.

5.3.3 Compute objective values using median This set of experiments uses the Median configuration. Since a large integer value is used for the planning time of any unsolvable

	Original	Median		
		Baseline	3-problems	Longer-macros
Depots	9.93	12.75	10.90	**12.90**
Driverlog	16.21	16.76	15.92	**16.92**
Rovers	17.17	**17.33**	16.88	**17.33**
Satellite	17.30	17.89	**18.57**	17.67
ZenoTravel	**18.96**	17.92	17.61	-

Table 7: IPC scores of the different ParamILS-inspired learning configurations for benchmark problems, when objectives are calculated by median.

	Original	Median		
		Baseline	3-problems	Longer-macros
Depots	11.58	16.85	13.55	**18.84**
Driverlog	19.01	**25.40**	22.53	24.76
Rovers	19.56	**24.12**	19.40	24.10
Satellite	20.35	**28.67**	25.71	27.93
ZenoTravel	26.79	**30.88**	29.24	-

Table 8: IPC scores of the different ParamILS-inspired learning configurations for all evaluating problems (including both benchmark and more complex problems), when objectives are calculated by median.

problems during the learning process, using mean planning time as objective values may lead to useful macro-actions being discarded if the choice of training problems was unfortunate. Therefore, median planning time may be a better way to calculate objective values, since it is not affected by extreme values. This set of experiments aims to assess whether using median instead of mean to compute objective values has any impact on the performance of the planner.

By comparing the Baseline columns in Table 7 and 8 with the Baseline columns in Table 5 and 6, this approach provides better results for almost all domains for both benchmark and the more complex problems. The only exception is the Rovers domain, where the performance for the more complex problems is slightly worse than that of the Baseline configuration. However, its performance is still much better than the original domain. Hence, on the whole, median run time is a better way to compute objective values.

5.3.4 Use a smaller number of training problems In this set of experiments, the 3 problems from the systematic learning approach are used instead of the 20 training problems. This set of experiments aims to assess whether the local search learning method is able to produce useful macro-actions with a small number of training problems. By comparing the 3-problems column and the Baseline column in Table 5 and 6, it can be seen that the learning outcome becomes significantly worse for the Rovers domain, where the performance of the learnt augmented domain is worse than that of the original domain. The Satellite and ZenoTravel domains show worse performance than the Baseline approach, but the performance is better than that of the original domain. The 3-problems approach works well on the Depots and Driverlog domains. Especially Depots, the scores are much higher than those of the other approaches which use 20 training problems. This is because the useful macro-actions for the Depots domain make some of the 20 training problems unsolvable with the afforded time and memory limits. Since the objective value of an augmented domain is calculated by mean run time, and a large integer value is used as the planning time when a training problem can not be solved, the useful macro-actions are discarded during the learning process due to large objective values. This issue is avoided when only 3 training problems are used, as the useful macro-actions are able to solve all of the training problems.

By comparing the 3-problems column and the Baseline column in Table 7 and 8, it can be seen that the 3-problems approach is always worse than the Baseline approach when the objectives are calculated by median, even for the Depots domain. This is because the median of just 3 planning times, rather than 20, is a poor indicator of overall performance.

5.3.5 Include longer macro-actions in the neighbourhood We observed with the systematic learning method that useful macro-actions may comprise more than two primitive actions. Previous sets of experiments here only include the initial set of macro-actions, which may not be enough to identify the most useful macro-actions.

By comparing the Baseline column and the Longer-macros column in Table 5 and 6, it can be seen that the approach provide significantly better macro-actions for the Driverlog and ZenoTravel domains. The other domains have similar or worse performance. By comparing the Median Baseline column and the Longer-macros column in Table 7 and 8, it can be seen that longer-macros only provides better macro-actions for the Depots domain when the objectives are calculated by median. No useful macro-actions are found for the ZenoTravel domain: the longer macros in the neighbourhood reduced the exploration of shorter ones, which were a better choice here.

6 Conclusions and Future Work

This paper shows that it is possible to learn macro-actions that can improve planner performance, even while excluded from search. We presented two ways of learning such macros, and were able to show improved performance in a range of benchmark domains. Our 'baseline' configurations of these offer good performance overall, with some further gains or losses to be had on a domain-by-domain basis as one varies the exact training configuration (hyper-parameters, training problems).

In future work, we will continue to explore how best to configure our local search learning process. For example, different termination criteria may be examined to assess their effect on the macro-actions learnt. We will also explore how effective our approach is with different heuristics, and with different benchmark domains. Another possible research direction would be to perform experiments using the macro-actions generated by other generators, such as MacroFF and Marvin, to see if they can help to improve the planner's performance when only used in heuristics.

References

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129: 5–33.

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.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Using Component Abstraction for Automatic Generation of Macro-Actions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 181–190.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11(4): 625–655.

Chrpa, L.; and Vallati, M. 2021. Planning with Critical Section Macros: Theory and Practice. *Journal of Artificial Intelligence Research*.

Coles, A.; and Smith, A. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research*, 28: 119–156.

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

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3(4): 251–288.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173: 503–535.

Hoffmann, J. 2001. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3): 57–62.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *J. Artif. Intell. Res. (JAIR)*, 36: 267–306.

Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1): 35–77.

Long, D.; and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *J. Artif. Intell. Res. (JAIR)*, 20: 1–59.

McDermott, D. 1996. A Heuristic Estimator for Means-Ends Analysis in Planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 142–149. AAAI Press.

Newton, M.; and Levine, J. 2007a. Evolving Macro-Actions for Planning. In *Proceedings of the Workshop on AI Planning and Learning held at ICAPS 2007*.

Newton, M.; and Levine, J. 2007b. Wizard: Suggesting Macro-Actions Comprehensively. In *Proceedings of the Doctoral Consortium held at ICAPS 2007*.

Newton, M.; Levine, J.; Fox, M.; and D., L. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 256–263.

Vallati, M.; Chrpa, L.; and Serina, I. 2020. MEvo: A framework for effective macro sets evolution. *Journal of Experimental and Theoretical Artificial Intelligence*, 32(4): 685–703.