Macro actions for structures

Alan Lindsay
University of Strathclyde, Glasgow
alan.lindsay.100@strath.ac.uk

Abstract

It is not surprising that structures underly many of the problems that we find interesting in planning. However, the planners that we develop are not always capable of acting on them as they increase in size. For example, the errors caused through relaxations in a heuristic can grow quickly when acting on a structure. Macro actions can help to compensate for heuristic error; however, researchers have investigated finite length macro actions limiting the benefit when the underlying problem is an arbitrary sized structure. In this work we design a specific set of arbitrary length macros, providing a vocabulary for acting on structures.

1 Introduction

The main focus of research in automated planning has investigated forward chaining approaches. The solution is constructed from beginning to end, which means the current state is always known. Over the years heuristics have developed and are now effective for planning in many problem domains (Hoffman and Nebel 2001). These heuristics have limitations that have motivated researchers to investigate approaches to supporting them with domain specific control knowledge (Coles and Smith 2007).

The language that is used to model planning problems is intended to be appropriate for expressing the solution. This has inspired researchers to investigate alternative languages that are appropriate for planning (Coles and Smith 2007; Botea et al. 2011). An approach that has received a lot of attention is to enhance the model with macro actions, which are formed by linking together sequences of actions (Coles and Smith 2007; Botea et al. 2011). This approach has the advantage that the reachable states are the same in both models; while also providing a new level for planning.

The process of generating macros in MacroFF (Botea et al. 2011) begins by dividing the domain structure into components: each component gathers together a group of objects that act together. The motivation is that tasks often require a number of preparation actions and these actions can be grouped into a useful abstraction. We have observed that planning problems often involve sequences of structure interaction that have arbitrary length. For example, in transportation problems, transporters are moved around a map redistributing packages and in structure building problems, blocks are joined together and removed to construct a structure.

In these problems behaviours are often repeated over a number of nodes in the structure. For example, a sequence of picking up and dropping off blocks in a Blocksworld problem. However, the sequences are acting towards a single higher level target, such as uncovering a specific block in a pile. This has motivated us to investigate a new problem language that allows the planner to make choices at this target level.

Our approach is to generate a collection of macro actions that provide specific behaviours on structures. We rely on the domain analysis tool TIM (Fox and Long 2001) and uncover two specific structure interaction problems: graph traversal and structure building. Many of the benchmark planning problems exhibit an aspect of one of these problems or both. We identify two key interactions on these structures and define the targets of these sequences. We then use training data to seed our macro actions.

We begin by presenting the background, we introduce the two key structure interactions and a selection of sequences that are important for solving these tasks. We continue by presenting our framework for generalising these sequences and presenting them as arbitrary macro actions. We survey the related work, evaluate the macros actions by providing them as options during search and make our conclusions and propose future work.

2 Background

A planning problem, $\mathbf{P} = \langle \mathbb{S}, \mathbb{A}, s_i, g \rangle$ can be represented as a set of states, $\mathbb{S}$, a set of actions, $\mathbb{A}$, an initial state, s_i, and a set of goal propositions, g. In this work, we represent states by a set of propositions, and the actions as three sets of propositions: the precondition (a_{PRE}) and the add (a_{ADD}) and delete (a_{DEL}) effects. We use $s' = a(s)$ as a function that returns the state after applying the action ($s' = (s \setminus a_{DEL}) \cup a_{ADD}$), defined for $a_{PRE} \subset s$.

A solution to a planning problem is a plan, $\pi = a_0, \ldots, a_{n-1}$, such that $s_1 = a_0(s_i) . \ldots . s_n = a_{n-1}(s_{n-1}) . g \subset s_n$. The plan is represented using the actions of the model; this means that the language of modelling and as a result planning is dictated by the desired plan language.

A common approach to discovering plans is to use heuristic guided search. Several heuristics have been developed that perform well on many problems. However, with any heuristic there are inevitably problems that have search spaces with local minima in the heuristic landscape. In particular, the depth of these local minima will determine how they impact on search. This has motivated researchers to investigate alternative problem model languages that reduce the depth of these local minima.

Macro actions provide a way of decoupling the language of the plan from the language used for planning. A macro action is an action sequence, $a = a_0, \dots, a_{n-1}$ that is applied as a single action. The state successor function is extended: $s' = a(s) = a_{n-1}(\dots a_0(s) \dots)$. These actions are generalised to macro operators by replacing the constants with free variables. A substitution, $\Theta = \{c_0 \leftarrow v_0, \dots, c_n \leftarrow v_n\}, \forall j \in [0, \dots, n]\ v_j \in V_{mop}$, replaces the variables in the macro with problem constants. Application of a macro operator is valid for a given substitution and state, if the action sequence after substitution can be applied to the state.

In (Coles and Smith 2007) macro operators are used to reduce the depth of local minima for planner FF (Hoffman and Nebel 2001). FF uses an enforced hill-climbing search, guided by the relaxed planning graph (RPG) heuristic: an estimate of distance to goal based on generating a plan for a relaxed problem. The relaxed problem is the problem remodelled so that the actions do not have delete effects. One derivative of the heuristic computation is a filtering function called the *helpful actions* (Hoffman and Nebel 2001).

Macro actions allow the planner to step through parts of the search space where it cannot plan effectively. A particular example of this is sequences of actions on structures: the error caused through relaxations in a heuristic can become exacerbated if acting on a structure. In (Hoffmann 2005) it is reported that 7 of the benchmark problems have RPG heuristic landscapes with arbitrarily deep local minima. A benefit of the approach presented in (Coles and Smith 2007) is that the macros are learned during planning. This means that their length is related to the problem structure. However, these learned macros are still of a fixed length. If a later part of search requires acting on a larger (or smaller) part of a structure then their assistance might be limited. In particular, the learning approach relies on the planner's ability to solve the problem the first time. One aspect of our research has investigated whether using arbitrary length macros is an effective approach to reducing the local minima.

3 Structures in planning

Structures in planning are used to represent various different relationships. These include: the spatial relationships between objects in the world, such as *can reach* or *can see*; and counters that limit the use of a resource, such as *fuel-level*. The domain analysis tool TIM uncovers implicit properties of the domain. It can be used to identify a wide variety of traversal problems (Fox and Long 2001) and we have used the properties to uncover some types of structure building problems.

In this section we introduce the two common structure interactions that occur in the planning benchmarks: graph traversal and structure building.

3.1 Graph traversal

Graph traversal problems involve moving an object through a constrained structure. This is particularly important for modelling problems with important spatial relationships. In these problems traversers move between different locations on a map. The location of the traverser constrains the actions that the traverser can perform. For example, in a transportation problem, a traverser picks up packages, but this can only be done at its current location.

A key aspect of the graph traversal problem is the action that moves objects between locations. The move action, (*moveAction* $t\ l\ l'$), has three parameters: the traverser, t, the traverser's current location, l, and the destination of the move, l'. The set of objects that can be moved by the move action are called the traversers and denoted $\mathbb{T}$. The set of positions that an object can be located are called the locations and are denoted $\mathbb{L}$.

The TIM analysis identifies transitions between relationships of the same type; in particular, an action that changes the location of a traverser from one location to another. There are two main traversal problems: transportation problems, for example, Logistics and Driverlog; and path opening problems, for example, Grid and Goldminer. Both of these can rely on a collection of carriables, $\mathbb{C}$, that are picked up by the traverser. For example, packages in transportation and keys in path opening problems.

3.2 Structure building

Structure building problems involve connecting similar objects together to form a particular structure. A key concept when planning in a structure building problem is whether the current structure is correct. This requires an understanding of reachability and the structure that is to be built.

A structure building problem is characterised by an attach action, (*attach* $o_1\ o_2$), and a detach action, (*detach* $o_1\ o_2$), each with two parameters: the two connected or connecting objects.

In the benchmark problems the structures are defined in goals; however, they could also exist as the precondition to an action. In this work we are interested in structures that are defined in the goal of the problem as these structures might be required to be of arbitrary size. Structures required to satisfy action preconditions have only to be of fixed size.

4 Sequences of structural interaction

We often have to perform the same task on an arbitrary set of the nodes of a structure. For example, in traversing a graph we might move between connected nodes; and in Blocksworld we might uncover a block by repeatedly unstacking blocks that are sitting on top of it. In this section we define two important structure interactions and establish a set of rules that determines the reason for the interaction. For example, a traverser is moved through the structure to pickup a package. Identifying these targets is an important step in our macro generation process.

4.1 Traverser reachability

A common task in traversal problems is moving the traverser to targets. To move a traverser between locations can require both move actions and additional enabling actions. We use a dependency graph to identify the thread of actions that contribute to the movement of traverser.

Dependency graph The achievers of an action's preconditions are made explicit in a dependency graph. We define this for a plan, $\pi = a_0, \ldots, a_n$, and problem model, $\mathbf{P} = \langle \mathbb{S}, \mathbb{A}, s_i, g \rangle$. The vertex set contains all of the actions (with suitable relabelling where duplicate actions exist in the list): $\mathbf{V} = \{a_j\}$. The edge set connects the action that achieves a proposition with actions that require the proposition to fire:

$$\mathbf{E} = \{a_i \to a_j | \exists p \ \max_i(p \in a_{i_{ADD}}) \wedge p \in a_{j_{PRE}} \wedge i < j)\}.$$

Move relevant actions For a given plan, π, and traversal behaviour we can define the set of move actions as $\mathbf{MoveActions}_\pi$ and the move relevant actions in the following set. These are simply the actions that are move actions, or actions that enable a move action:

$$\begin{aligned} \mathbf{MoveRelevantActions}_\pi = \{a \in \pi \mid \\ \exists (a_0, a_1) \in E, \ldots, (a_{n-1}, a_n) \in E \\ a_n \in \mathbf{MoveActions}_\pi\} \end{aligned}$$

Targets The thread of move relevant actions can be split into sequences that achieve a particular target. For example, in a transportation problem each package introduces two definite targets: its current location and its destination. However, there are some targets that are not as clearly defined. For example, in Goldminer problems we must drop the laser in order to pick up the bomb. The laser will have served its purpose and has no goal: it does not matter where it is dropped and so is not a target.

We define the following rules for breaking traverser threads into sequences:

1. Move to goal : a goal achieving action is enabled by a move action.
2. Pickup carriable (either a package or enabler)
3. Drop-off package (not an enabler - unless achieving its goal)
4. End of an opening episode (where the current enablers were sufficient for the traversal task)

These four rules provide a strategy for breaking up any thread of traverser relevant actions. An important motivation of moving is to support achieving a goal. This is common to each of the categories of traversal problems. Similarly it is usually necessary to move to pick up objects, either to distribute or to aid traversal. However, we have observed that the position an object is dropped off at is usually only important for transportation problems. Rule 1 partially covers situations where this is not true. The final rule is relevant for path opening problems. The sequence should not be separated at each opening, because there may be several in a row. Instead the sequence should be broken by the last opening action achieved using the current enablers.

In addition, once split into a sequence, an action that does not enable later parts of the sequence are pruned. We discuss our assumptions below.

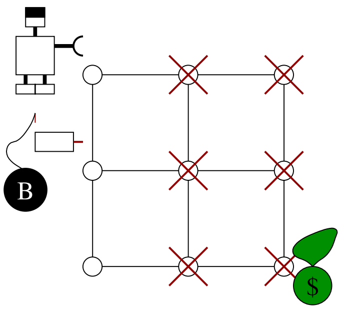

Figure 1: $\mathbf{P}_{Example}$, an example problem. The objects are named: robot, r; laser, l; bomb, b; and the location in row, i, and column, j, is called l_{ij}. The goal is to pick up the gold.

We use the example illustrated in Figure 3 to make the application of the rules more clear. The purpose of Goldminer problems is to open up a path so that the robot can pick up the gold. The only action that is not relevant to moving is the final pick up action. There are usually four targets in Goldminer problems: pickup the laser; fire to one location from the gold; pickup the bomb; pickup the gold. The problem in Figure 3 can be solved with the following steps:

1. move r l_{00} l_{10}
2. pickup r l $l_{10}{}^2$
3. move r l_{10} l_{20}
4. fire r l l_{20} $l_{21}{}^4$
5. drop r l l_{20}
6. move r l_{20} l_{10}
7. pickup r b $l_{10}{}^2$
8. move r l_{10} l_{20}
9. move r l_{20} l_{21}
10. blow r b l_{21} l_{22}
11. move r l_{21} $l_{22}{}^1$
12. pickGold r l_{22}

The target actions are marked in red with the breaking rule index in superscript. Action 5 will be removed from its sequence as it does not enable anything in the chain. As the grid grows in size the number of actions between targets increases. For example, the laser might be located four steps away from the robot's starting location. The length of these sequences are determined by the problem instance and are therefore of arbitrary size. Figure 2 illustrates the structure that is navigated for the robot to reach a square away from the gold. The general form of the resulting sequence is:

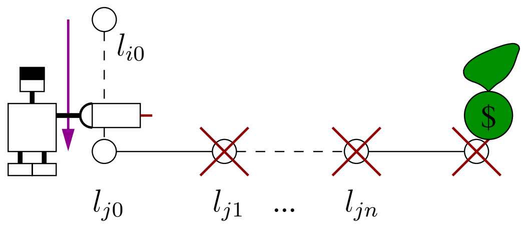

Figure 2: The robot moves through a sequence of moves then fire move pairs.

$$\text{move } r\ l_{i0}\ l_{(i+1)0};$$
$$\ldots$$
$$\text{move } r\ l_{(j-1)0}\ l_{j0};$$
$$\text{fire } r\ l\ l_{j0}\ l_{j1};\ \text{move } r\ l_{j0}\ l_{j1};$$
$$\ldots$$
$$\text{fire } r\ l\ l_{j(n-1)}\ l_{jn};\ \text{move } r\ l_{j(n-1)}\ l_{jn}$$

4.2 Unstacking a block

The most common structure building problem in planning is the stacking problem. We have limited our study to this form of structure building problem. The structures that are acted on in these problems are stacks and the planner is limited to interacting with the structures from its top element. We assume there is a storage space with sufficient room, for example a table. A necessary sequence acted on these stacks is uncovering a block from a structure. Uncovering blocks requires a chain of actions that iteratively removes blocks from a structure. Each of these removal steps might require several actions: if the movement of a block is separated into distinct actions, or if the actions need enabled.

The relevant actions can be extracted, selecting those actions that enable the removal step actions (for example the pickup and put-on-table actions in Blocksworld), in a similar method as for traversal actions. The target for an uncovering block problem is that a particular block is free to be picked-up. We use a structure to determine whether a block was an important target.

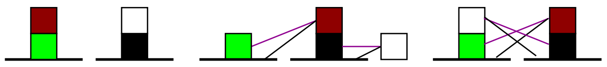
Figure 3: Example structure, with old-connections in black and below-connections in purple. When white is put on green we examine the stack old-connections and find that red is on black.

The structure updates two distinct sets of connections that record the previous stack of a moved block. When a block is detached from a stack a *below-connection* is made with each block underneath it. When the block is attached to a new stack an *old-connection* is made with its old stack. If a moved block is already connected then we do not update the connections. If it has been attached to an intermediary stack then this could have been achieved using the table. We use this structure to determine whether uncovering a particular block was the target.

We define the following rule to divide the thread of stack interactions into sequences:

1. A block, b, is a target if: b is being attached; b will not be removed; and each of the blocks that have old-connections to b's stack are in the final state not in stacks above blocks that are below-connected to b.

This last property states that if a block was underneath b and will be underneath a block that has been moved from b's stack then it was the reason that b was moved. Therefore unstacking b was not a target. This allows us to break the thread into single sequences that achieve important target. In the next section we describe a strategy for generating macro actions from these sequences.

4.3 Assumptions

In this section we have made two assumptions about the traversal threads of interaction:

- single independent subgoals that require a traverser at a particular location are identified by the targets;
- plans demonstrate a single traversing thread.

The first assumption means that any action in a sequence is contributing to moving the traverser to a specific location. The intention is that the target rules break the chain at the important subgoals. If important subgoals are achieved during a chain then the generalised representation of this chain will not enforce this. We have generated the rules based on a survey of the benchmark domains and defined them in terms of general properties of these types of problem. The second assumption means that we do not require to unpick a thread of movements from a sequence. This has the side effect that underlying the macros is the assumption that traversers are independent of each other. These assumptions hold in most benchmark problems. We believe that separating out the threads of traverser moves is possible and pose it as future work.

5 Arbitrary macro actions

In this section we present our representation for arbitrary macro actions. We use macro operators as the building blocks that capture sequences of actions. We generate the set of all states reachable through application of these macro operators.

In this section we define our training data representation, our approach to generalising the sequences we defined in Section 4 and how these are used to generate macro actions.

5.1 Training data

We use training data to guide the action sequences that we consider. The domain conventions often greatly reduce the possible chains of actions. We want to exploit this to ensure the number of macros remains as small as possible. We also appeal to the assumptions made of the sequences.

In this section we rely on training data in the form of problem plan pairs:

$$\langle(|\pi_0|, \mathbf{P}_0), \ldots, (|\pi_n|, \mathbf{P}_n)\rangle.$$

For example, the problem, $\mathbf{P}_{Example}$, and the plan presented in Section 4 could act as a training data pair.

5.2 Bags of macro operators

The use of macro actions is effective because in many problems short sequences of actions combine to perform a single task. We observe that when acting on a structure we often aim to perform a single task at each of a series of nodes. The precise details of how this tasks is achieved might vary: for example, some nodes might require preparation. Our approach is to observe each action sequence that is used to achieve a task and group them into bags:

Definition 5.1 *A bag of macro operators is a set:* $\mathbf{macroBag} = mop_0, \ldots, mop_n$ *over distinct variable sets:* $\forall i, j \in [0, \ldots, n]\ i \neq j \implies \forall v \in \mathbf{V}_{mop_i}\ v \notin \mathbf{V}_{mop_j}$.

For example, consider the target of opening the location next to the gold in Goldminer. At each individual node we can focus on two possible moves: move the robot into an open square; shoot at rock and move into the square. Alternative action choices, such as putting the laser down, are not important to the current target. These two alternatives each make a single movement. They can be represented by macro operators and we can define a bag **fireMoveBag**:

- (move ?robby ?l1 ?l2);
- (fire ?robby ?laser ?l1 ?l2), (move ?robby ?l1 ?l2).

The two fire actions share variables with the subsequent move actions. This means that the operators perform a single task of moving the robot to an adjacent square, perhaps opening the square first.

In general we can construct bags of operators that support all of the behaviours that we may require at a particular node. If this is done in the context of a particular target, then we can reduce the possible chunks to a useful subset.

Chunking a sequence In Section 4 we defined an approach to generating sequences. Each sequence includes several examples of the actions that are applied at each node. We now use these sequences to generate macro bags.

Each interaction action with its enablers is an example of a possible approach to interacting with the structure. We call each of these possible interaction situations a *chunk*. A chunk is a subsequence that extends from either the beginning of a sequence, or the action after an interaction, to the next interaction action. For example, the sequence we presented in Subsection 4.1 breaks into chunks:

$$
\begin{gathered}
(\text{move } r\ l_{i0}\ l_{(i+1)0}) \\
\ldots \\
(\text{move } r\ l_{(j-1)0}\ l_{j0}) \\
(\text{fire } r\ l\ l_{j0}\ l_{j1};\ \text{move } r\ l_{j0}\ l_{j1}) \\
\ldots \\
(\text{fire } r\ l\ l_{j(n-1)}\ l_{jn};\ \text{move } r\ l_{j(n-1)}\ l_{jn})
\end{gathered}
$$

Each chunk is generalised by replacing the constants with variables. Each chunk defines a macro operator and we construct a bag from the macros of a sequence. In the above example the chunks collapse into **fireMoveBag**. Informally, for unique chunks, $c_0, \ldots, c_n$ in a sequence, we generate an action with regular expression: $(c_0|\ldots|c_n)*$. We make this formal below.

This approach of combining the macros collapses any constraints in the order of execution. Part of our future work will examine the effect of maintaining a graph that records when a chunk is observed after another chunk. This graph could then be used to restrict the valid bag expansions.

Reachability graph for bags The reachability of a bag can be represented as a graph. The nodes are states and the edges are the instantiations of the macro operators in the bag. There are binding constraints associated with the bags that ensure the sequence contribute to a single interaction. These are that the same traverser is moved, or that the same stack is popped. The edge weights are a count of the number of actions in the macro. A target is reachable from the current state with a bag if there is a path in the reachability graph from the current state to a state that satisfies the target.

5.3 The macros

A set of macro actions is computed at each state. The reachability graph is computed. For each target discovered in the reachability graph a macro action is defined. The sequence of enablers and moves made to reach the target are wrapped up in a single option for the planner.

6 Filtering the macros

The set of macro actions that we have generated in Section 5 can be large (intractable). Adding macro actions does not increase the total number of states in the model; however, the branching factor quickly increases. In this section we consider three approaches to reducing the set of macros.

6.1 Target significance

The graph that we have proposed is potentially large. We aim to allow the planner to achieve specific targets directly. However, the macro set that we proposed is likely to have many states that achieve the same target; illustrated in Figure 4. Moreover there are many paths to each target, creating a significant search space. We have observed that often any path that achieves a target can be extended to the reach the same set of targets. We limit this set to a single macro per target (4b) and expand a single state from each target (4c).

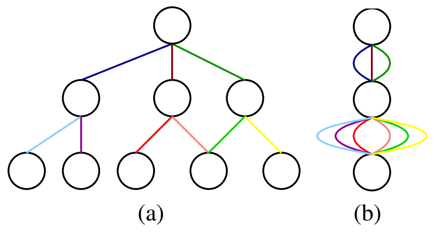

Figure 4: Example of bag expansion using 4a state indexing; 4b the result when it is projected into the target space; and 4c the expansion using target significance.

We use the target as the index during expansion. The graph is generated in a breadth first search and the first achiever of a target is selected. A chain is only continued if it achieves a target that has not been discovered already. We define the property *target significance* that holds if the target of the sequence discriminates enough, so that if any two states have the same target, then they are extendable to the same targets.

We have experimented with using training data to determine whether the chains are target significant; however, our assumptions and the selection of rules we defined in Section 4 lead to target significant chains in the domains we have experimented with.

6.2 Target identification

We have generated a set of macros that allow the planner to perform structure interaction in single steps. However, there are many nodes in the structure that are not interesting. For example, a location in a transportation problem with no goals or packages at it. In particular, we have already defined a set of rules that determine the targets for the structure problems. If these rules do not require knowledge of future steps then it is possible to use the rules to determine whether a macro action concludes at a target node.

We define the property *satisfies rule* that holds for a triple, (s, a, g), if the macro action, a, current state, s, and goal, g, satisfy one of the rules. The macro actions that have been generated are only added to the problem model if this property holds. An example transportation problem has a truck, t, package p, and two locations, l_1, l_2, with $s = (\text{in } p\ t) \wedge (\text{at } t\ l_1)$ and $g = (\text{at } p\ l_2)$. There are two macros: (move $t\ l_1\ l_2$) and (move $t\ l_1\ l_1$). The package goal at l_2 means that the satisfies rules property holds for the former macro through rules 1 and 3 and is retained. However, the latter macro does not satisfy the property and this macro is pruned.

The rule defined for structure building requires knowledge of the future and could not be used for pruning. It is clear that this approach can remove useful targets, for example, putting a package down at a hub location. However, these actions are still possible by selecting problem model actions.

6.3 Forcing the use of macros

A final strategy we have investigated is forcing the use of the macro actions. This can be achieved by identifying all of the actions that are part of the macro sequences and preventing the planner from selecting these actions unless within a macro action. The actions removed for the **fireMoveBag** are fire and move.

7 Experiments

Our experiments aim to investigate the suitability of structure interaction macros as a support for heuristic planning and whether arbitrary length macros can be part of an efficient solution to the planning problem. We present a preliminary investigation that answers these questions using the results from two domains.

The basic search (**Basic**) uses an enforced best-first search, informed using the relaxed-plan heuristic (Hoffman and Nebel 2001). A breadth first expansion is used on reaching local minima[1] that expands in layers until an improving node is discovered. We use the helpful actions filter first and restart if no improvement. This is similar to FF (Hoffman and Nebel 2001); however, we select the best discovered neighbour and not the first improving. Our solution is developed within JavaFF (Coles et al. 2008), which has implications for its efficiency.

For each domain we generate a set of macro actions using the approach presented in Section 5; we then test these actions by incorporating the macros into our search strategy in two different ways: throughout search and to escape local minima. In our experiment we compare these two configurations with the basic search approach.

An interesting aspect that we investigate is whether the search time increases due to the increased branching factor. We can determine this by comparing the search time when using the macros either throughout search or for escaping local minima with basic search.

The heuristic is not informed with the length of the macro actions. This lack of information could have a large impact on the quality of solutions. It is therefore interesting to examine the quality of solutions in comparison with basic search.

7.1 Macros in search (SeqFF)

The first approach is to allow the planner to select from either the described actions or the macros at each search step. During best-first search the helpful actions are generated. The reachability graph is computed for each bag at each state. The target states are added to the states computed by applying each helpful action to the current state. The state with the lowest heuristic score is selected.

When search finds a local minimum we use a modified approach. We enforce that a macro action cannot be applied directly after another and we force the use of the macro using the above approach. We have also used this configuration with no macro force in Goldminer for comparison (**No-Force**). All of the states are added to the open list and expanded in the next layer. If the current layer contains states with lower heuristic than the local minimum then the best of these states is selected and best-first restarts from this state.

7.2 Macros for escaping local minima (MinimaEscape)

The second approach is to use the helpful actions during the hill-climbing search, but allow the planner to use either the described actions and the macros during the local minima escaping expansion. This approach was used in (Coles and Smith 2007); however, the macro actions had finite length. This configuration is interesting because we can investigate whether structures impact on the ability of search to escape from local minima.

7.3 Results

We have used these approaches to solve problems from two benchmark domains. In this study we have only investigated our approach on problems with traversal subproblems; however, we do predict expected results for structure building problems.

Goldminer The first domain that we apply our strategy to is the Goldminer domain. This domain was not surveyed in (Hoffmann 2005); however, it is clear that Goldminer problems contain arbitrary local minima for the RPG heuristic. This is because at the stage where the robot has cut a path to one step from the goal with the laser, the heuristic has estimated that all that is required is for the robot to fire the gold square, drop the laser and run in to claim the gold.

[1]Actually at a plateau or local minima.

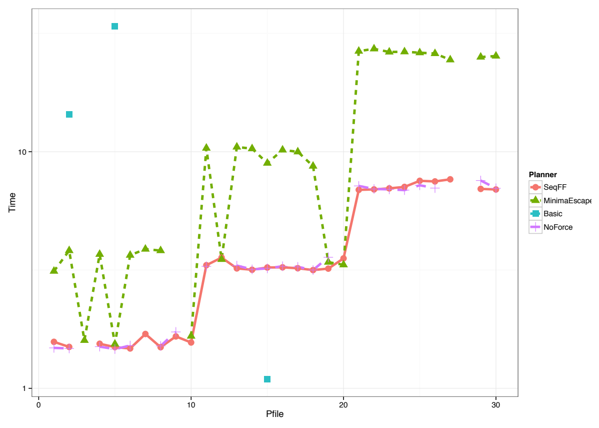

Figure 5: Time results for Goldminer domain.

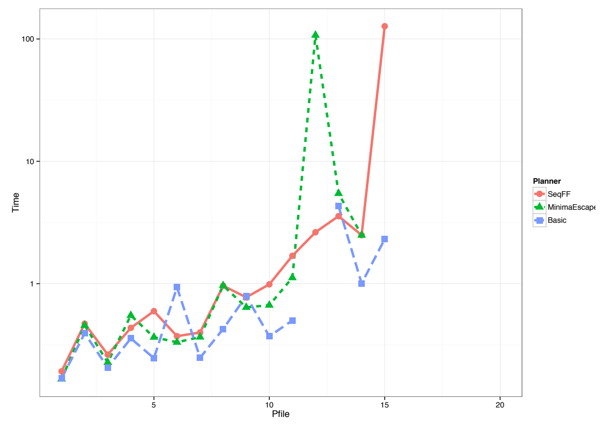

Figure 7: Time results for Driverlog domain.

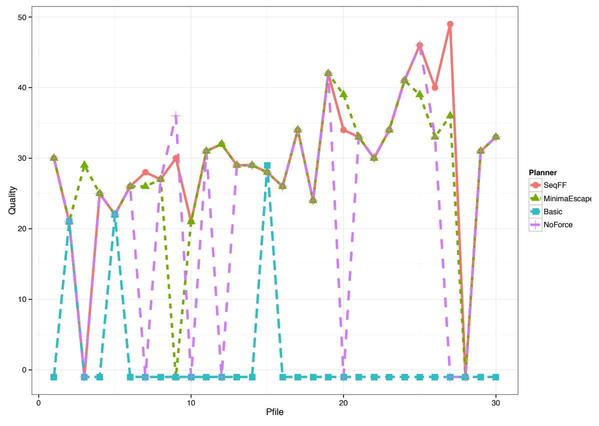

Figure 6: Quality results for Goldminer domain.

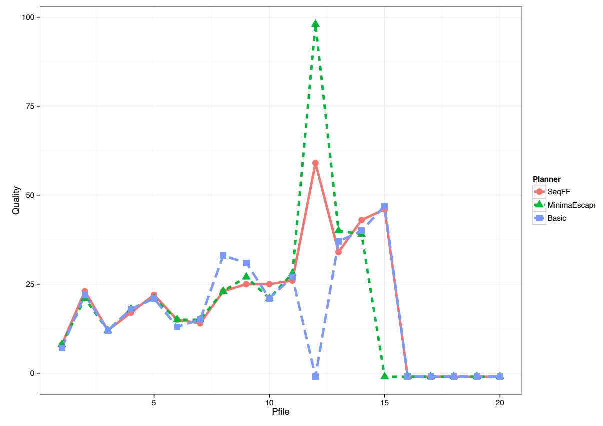

Figure 8: Quality results for Driverlog domain.

However, the robot needs to travel a number of steps (determined by the initial state) to pickup the bomb. Moreover the state space of Goldminer is very large and the breadth first expansion is very expensive. If the grid is size n and the bomb is positioned m steps away from the path that was cut out to the gold then the local minima will be of depth $2(n-1)+2m+2$. In contrast, the macro actions that we have proposed reduce this local minima to a depth of 4.

The generated bags were:

- move*
- (move | fire; move)*
- (move | detonate; move)*

The time and quality plots are presented for 30 problems distributed as part of the first learning competition (Fern, Khardon, and Tadepalli 2011). These problems have been remodelled to remove the conditional effects and implicit objects. The time results are in seconds and plotted on a log-scale axis. The quality results are in plan steps. Failed execution is plotted as a missed point on the time axis and as -1 on the quality plot.

Driverlog We have chosen Driverlog as it has an underlying graph structure and is one of the domains reported to have arbitrary local minima (Hoffmann 2005). We have plotted the time, Figure 7 and quality, Figure 8 results on the 20 problems generated for the third planning competition (Long and Fox 2003). The plots are presented in the same way as for Goldminer.

The generated bags were:

- drive-truck*
- walk*

7.4 Comparison

The results for Goldminer demonstrate that structure based macro actions can be an effective support for the RPG heuristic. The base line search solves 3 problem, while both approaches using macro actions solve 28 of the 30 problems. The time results demonstrate that using the macro actions during search can make substantial improvements on time over restricting use to the minima. However, more frequent use of macros has a negative effect on plan quality. The missed problems are caused by the heuristic's favour towards using the bomb: it gets a free hand and an open square. In most cases a plateau is entered and the benefit of the macro actions are discovered. This would be solved if the macros were part of the heuristic computation. In this experiment the macro force filter is acting to prevent the selection of the detonate action. It is therefore important to note that the macros are not solving the problem of undetected dead-ends in Goldminer.

As can be observed from the Driverlog plots, on most problems the macros make little difference, on some the macros perform worse. One reason for this is that the depth of the local minima depends on the difference in distance between the drivers' map and the trucks' (Hoffmann 2005) and in the generated problems there are few examples where this distance is large. This is because the graphs are densely connected. In particular, the aspects of the problem where macros might have helped are too small to make the overhead worthwhile.

The macros that we generate can provide an important option if the heuristic has local minima caused by interaction with a structure. In particular, if the macros reduce the depth of the minima and the filtering by target significance reduces the states that need explored.

Structure building Although we have not examined the behaviour of our macros with structure building problems we have predicted the results for the Blocksworld domain. In Blocksworld problems a tower that must be dismantled to uncover a misplaced block can satisfy some part of the goal. To uncover the block we will have to destroy these goals, resulting in an arbitrary minima (Hoffmann 2005). We predict that the macros presented here would reduce the size of the minima. However, there can still be arbitrary local minima under this enhancement.

In (Hoffmann 2005) it is shown that enhancing the problem model with the actions pickup-stack, unstack-stack and unstack-putdown then the heuristic has no local minima. It seems that an approach to generating finite macros would be appropriate for supporting the RPG in this domain.

8 Related work

There are several related works in macro learning. In (Coles and Smith 2007) the macros are learned to escape local minima and in (Newton et al. 2007) macros are extracted from contiguous subsequences of plans. Another key motivation came from (Botea et al. 2011): macros are generated from problem structures and also through analysing causal relationships in the actions. In our work suitable macros are selected by domain analysis; whereas in these works the selection of macros is biased by performance. The macros in these works are of fixed length.

Repeating sequences have been investigated in the context of generalised plans (Winner and Veloso 2007) and rule based policies (Lindsay, Fox, and Long 2009).

9 Conclusion and future work

We have presented a novel approach to generating and representing arbitrary length macro actions. We have defined the property of target significance and using this we have demonstrated that the model enhancements can be provided efficiently. We have developed a method for generating macros actions for traversal and stacking problems. We have demonstrated that when a problem has certain underlying structures the macro actions can be used with the RPG heuristic and improve its performance in terms of problems covered. However, the sequences that we have proposed are not guaranteed to remove the weakness in the RPG heuristic. Our future work includes: generalising the generation process to many traversers; investigating with sequences designed to assist the heuristic; and exploiting the macro actions with other heuristics, including rule based policies.

References

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2011. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *CoRR* abs/1109.2154.

Coles, A. I., and Smith, A. J. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Teaching forward-chaining planning with javaff. In *Colloquium on AI Education, Twenty-Third AAAI Conference on Artificial Intelligence*.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 84(1-2):81–107.

Fox, M., and Long, D. 2001. Hybrid STAN: Identifying and managing combinatorial sub-problems in planning. In *Proceedings of 17th International Joint Conference on AI*, 445–452. Morgan Kaufmann Publishers.

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

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Lindsay, A.; Fox, M.; and Long, D. 2009. Lifting the limitations in a rule-based policy language. In *The 22nd International Florida Artificial Research Society Conference*.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of AI Research* 20:1–59.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 07)*.

Winner, E., and Veloso, M. 2007. LoopDISTILL: Learning looping domain-specific planners from example plans. In *International Conference on Automated Planning and Scheduling, Workshop on Artificial Intelligence Planning and Learning*.