Integrating landmarks in partial order planners

Bram Ridder and **Derek Long**
University of Strathclyde
Glasgow, UK

Abstract

In this paper we continue a line of research which focusses on reviving partial order planning. We focus on utilizing landmarks (Porteous and Sebastia 2000) to split a planning problem into a set of subproblems. Our goal is to work towards a planner which solely uses lifted actions and this work shows some progress. We use the planning problems presented at the 3rd International Planning Competition (Long and Fox 2003) and compare the results of our approaches to the original VHPOP using lifted actions.

Introduction

In this work we continue a line of work that began with the revival of partial order planning by a planner called RePOP (Nguyen and Kambhampati 2001), which challenged the pessimism about the performance of partial order planners by integrating state-of-the-art state-space planning techniques and showed dramatic improvements. More recent work saw VHPOP (Younes and Simmons 2003) entering the IPC-3 (Long and Fox 2003) planning competition, again making use of state-of-the-art improvements in state-space planning, and which was able to tackle temporal domains winning the best newcomer prize. However, neither of these approaches use lifted actions in their tests, because no effective method has yet been found to take advantage over grounded actions. To continue the line of work in partial order planning we set out to evaluate the latest advances in planning and incorporate and improve upon these techniques in a partial order planning context. Our ultimate goal is to plan efficiently with lifted actions and, ultimately, to forego grounding altogether.

In this work we focus on integrating landmarks into the latest version of VHPOP and utilize this information to split the planning problem into smaller subproblems that are (we hope) easier to solve than the original problem. Our hope is that this renewed effort will revive the interest in partial order planning and present a showcase for its merits. In the first section we present the formalism used in the remainder of the paper. Next we present a way to split up the planning problem into multiple subproblems and finally we will discuss the search process and results.

Partial order planners

A (partial) plan can be represented by a tuple $\langle A, L, O, B, G \rangle$, where A is a set of operators, L a set of causal links, O a set of ordering constraints defining a partial order on the set A, B a set of binding constraints on the action parameters, and G the set of open conditions to be satisfied. Each action a is an instance of some operator A in the planning domain and a plan can contain multiple instances of the same operator. A causal link, $a_i \xrightarrow{q} a_j$ represents a commitment by the planner that precondition q of action a_j is to be fulfilled by an effect of action a_i.

When given a planning problem, an initial partial plan is generated by creating two additional actions: a_0 which contains as effects all literals in the initial state s_0 and a_∞ which has as preconditions the set of goal literals s_g. The partial plan is now generated by adding these two actions and by ordering a_0 before a_∞: $\langle \{a_0, a_\infty\}, \emptyset, \{a_0 \prec a_\infty\}, \emptyset \rangle$.

A refinement planner works by adding elements to a plan in order to remove flaws in the plan. A flaw can either be an open condition $\xrightarrow{q} a_i$, which represents precondition q of an action a_i which is not yet supported by another action, or an unsafe link (or threat) $a_i \xrightarrow{q} a_j$, whose condition q can unify with the negation of an effect of an action a_k that could possibly be ordered between a_i and a_j. There are 3 different solutions to this problem: 1) Either a_k is ordered before a_i (demotion); 2) a_k is ordered after a_j (promotion); 3) or a binding constraint is introduced so that the effect of a_k cannot unify with q (separation).

During the planning process a partial order planner keeps track of its plan-space in the set P. During every iteration a plan $p \in P$ is selected and then a flaw is selected to be resolved in p. All possible refinements resolving the flaw are returned and added to P, until either P is empty (denoting that no solution exists for the problem) or a plan without flaws is found (a solution).

SAS^+ formalism

A concise SAS^+ representation of a planning task can be generated from a typical PDDL representation automatically (Helmert 2009).

Definition 1 *A SAS^+ planning task is a tuple $\Pi = \langle V, O, s_0, s_g \rangle$ where:*

- *V is a finite set of multi-valued* state variables, *each with a finite domain D_v. A fact is a pair $\langle v, d\rangle$ (also written $v \mapsto d$), where $v \in V$ and $d \in D_v$. A partial variable assignment s is a set of facts, each with a different variable. A state is a partial variable assignment for all variables $v \in V$.*
- *O is a set of operators, where an operator $o \in O$ is a tuple $\langle name, prec, effects\rangle$ of partial variable assignments.*
- *s_0 is a state called the initial state.*
- *s_g is a partial variable assignment called the goal.*

An operator $o = \langle name, prec, effects\rangle \in O$ is applicable in state s if $prec \subseteq s$. In that case, it can be applied to s, which produces the state s' with $s'(v) = effects(v)$ where $effects(v)$ is defined and $s'(v) = s(v)$ otherwise. We write $s[o]$ for s'. For operator sequences $\pi = \langle o_1, \dots, o_n\rangle$, we write $s[\pi]$ for $s[o_1]\dots[o_n]$ (only defined if each operator is applicable in the respective state). The operator sequence π is a plan iff $s_g \subseteq s_0[\pi]$.

Integrating landmarks into VHPOP

The last partial order planner to enter the competition was VHPOP (Younes and Simmons 2003) at ICAPS'03 (Long and Fox 2003). Unfortunately, it was not able to perform competitively with state-space planners such as FF (Hoffmann and Nebel 2001). In an attempt to bridge the gap in performance between partial order planners and state-space planners we explore the option of exploiting landmark information. In this paper we discuss an approach which utilizes landmarks to split the problem into several subproblems. Previous works – like STeLLa (Sebastia, Onaindia, and Marzal 2006) – have attempted similar approaches using disjunctive landmarks, but our work makes no attempt to present the planner with a consistent set of landmarks to plan from. Instead we allow inconsistencies in every subproblem – i.e. allow multiple landmarks which define distinct values for a state variable – and employ lifted actions to handle these inconsistencies as we will explain below.

Splitting up the problem

In this section we describe the method employed to split up the planning problem into subproblems, using landmarks. One of the earliest papers on landmarks detailed a way to split planning problems using landmarks (Hoffmann, Porteous, and Sebastia 2004)), by successively planning to the "nearest" landmark until all are visited. We propose a different approach. First we discuss the process we used to derive landmarks and the modifications we make in comparison with earlier approaches. Next we explain how the *landmark graph* is used to split the problem up into consecutive planning problems and finally we report on the heuristics used and results obtained.

Deriving landmarks

Whereas state-space planners can successively plan to the "nearest" landmark, using plan-space planning we do not have the same option as we lack an explicit state definition. Furthermore, landmark orderings derived using the LM^{RPG} (Richter, Helmert, and Westphal 2008) algorithm are not sound which can distort the planning process and can cause the planning process to fail on a task, even though the underlying planning process is complete.

For this reason we use the landmark generation process derived by Richter, Helmert and Westphal (2008) which produces sound orderings, leading to shorter plans and an improved success rate, compared to LM^{RPG}, when applied to the same planners. As already observed, the process we use to generate the landmark layers is quite similar to the approach adopted by STeLLa (Sebastia, Onaindia, and Marzal 2006).

Determining the ordering of landmarks

Given a landmark graph $G = \langle L, E\rangle$, where every vertex $l \in L$ represents a landmark and every edge $e \in E$ is labelled with the type of ordering between its two vertexes: $e = \langle from, to, edgetype\rangle$. The landmark generation graph can be split up into separate landmark layers by iteratively grouping all landmarks that have no incoming edges. We start by labelling all landmarks as active. The first set of landmarks is the initial state. After every iteration we label all discovered landmarks as inactive and repeat the procedure, until all landmarks have been marked as inactive. For every landmark we denote the iteration number at which it was made inactive, which we refer to as the *layer number*, this procedure ensures that, if landmark l_1 is ordered before l_2 in the landmark generation graph, then the layer number of l_1 is less than the layer number of l_2. Relating this work back to STeLLa, this process guarantees the ordering property.

Creating the landmark layers

Now we propose our stratification technique. Given a set of pairs of landmarks and their respective *layer number* $H = \langle l, n\rangle : l \in L$; and $n \in \mathbb{N}$, we divide these into consecutive subsets $X_1, X_2, \dots, X_n$, such that every subset defines a state. The number of subsets is equal to the the highest layer number. For the remainder of the discussion we will add a notion of directionality: because we are doing a goal-directed search, we say that the first subset is the goal set and the last is the initial state.

Definition 2 *A landmark $l \in L$ defines a variable $v \in V$ if either of the following properties hold:*

- *l is not disjunctive and the value of $l \in D_v$, or*
- *l defines two or more distinct values for v.*

Definition 3 *A set of landmarks L defines a state if $\forall_{v \in V} \exists_{l \in L} | l$ defines v.*

Definition 4 *The* minimal landmark layer *at layer j is defined as the subset $X_j = \bigcup_{v \in V} X_j^v$. The set of values of every state variable $v \in V$ in X_j is defined as:*

$$X_j^v = \begin{cases} \{l : \langle l, j\rangle \in H\} & \textit{if } l \textit{ defines } v \\ X_{j+1}^v & \textit{otherwise} \end{cases}$$

This definition ignores all landmarks that do not define a variable, but it gives us a stratification of the planning problem. However, the choices represented by disjunctive landmarks are an essential part of planning, especially as more resources are available to accomplish a task. Therefor we store all landmarks we ignored in the above definition in another set, the *landmark action*.

Definition 5 *Given a* minimal landmark layer *at layer j X_j, the* landmark action *at the same layer is defined as the subset Y_j. The set of values of every state variable $v \in V$ in Y_j for which the set X_j was found at layer i is defined as:*

$$Y_j^v = \bigcup_{i > x \geq j} \{l : \langle l, x\rangle \in H \wedge l \textit{ does not define } v\}$$

We can now define the successive *landmark layers* as $Z_1, Z_2, \ldots, Z_n$, where $Z_i = \langle X_i, Y_i\rangle$.

Search process

Given an initial plan $P : \langle A, L, O, B\rangle$, we define the first subproblem as: $P_1 = \langle\{a_\infty, X_1, Y_1\}, \emptyset, \{X_1 \prec Y_1, X_1 \prec a_\infty, Y_1 \prec a_\infty\}, \emptyset\rangle$. To estimate the number of steps that must still be completed from the current *landmark layer* to the initial state, we apply the FF heuristic h_{ff}. Since h_{ff} uses the RPG to derive its heuristic we can deal with goal literals which define distinct values for state variables as it simply accumulates all positive effects. To solve P we use VHPOP with lifted actions.

A number of changes have been made to make better use of the lifted action representation. For every lifted action we define a domain for each of its variables and allow bindings between these variables and a set of atoms. When disjunctive landmarks are present in the *minimal landmark layer* or the *landmark action* causal links can be constructed which bind variables to a subset of effects which support an open condition. For example, if we have an open condition *(at package1 s1)* which is satisfied by the action *(DROP-PACKAGE package1 s1 ?var1)* and we have multiple supporters for the open condition *(at s1 ?var1)*, we bind the variable *?var1* to that set, e.g. *?var1* $\in \{truck1, truck2, \ldots, truckn\}$. Threats in a partial plan can be resolved by adding restrictions to the domains of the variables (separation), when a domain becomes empty the partial plan cannot be further refined and we need to backtrack.

Transitions to next subproblems

Once a subproblem is solved, we move on to the next subproblem. To do this, we first remove the *landmark action* and *minimal landmark layer* and all causal links it supports. Next we insert the next set $\{X_i, Y_i\}$, with the appropriate orderings $\{X_i \prec Y_i, X_1 \prec a_\infty, Y_1 \prec a_\infty\}$ as before. A subproblem is solved if it contains no flaws, note that we do not force variables to have a single value in its domain. This allows us to refine the domains of the variables as we solve the subsequent subproblems and only need to restrict the domains when we encounter threats. For example, in Figure 1 the subproblem is solved by adding the binding: *?var1* $\in$ $\{$*truck1*, *truck2*, *truck3*$\}$ as we do not force the planner to select one over the other at this point. Rather, we have the advantage of postponing this decision when we can make a more informed decision.

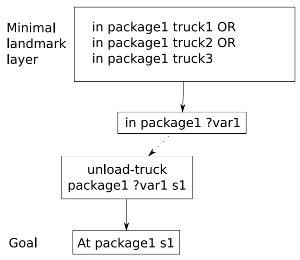

Figure 1: We solve this subproblem by binding *?var1* to $\{$*truck1*, *truck2*, *truck3*$\}$

In a similar fashion: if we have a threat in our plan with multiple refinements to resolve this threat, we allow them to persist in a solution to a subproblem and only refine them when absolutely necessary.

SAS$^+$ mutex relations

The usage of the *SAS*$^+$ representation allows us to derive more mutex threats. VHPOP only detects a mutex relation when a delete effect threatens a causal link, however consider the following: when a causal link is created from the initial state to the goal state supporting a value $v_1 \in V$. This means that no other value of the same variable can be supported by any causal link. However, VHPOP allows these causal links to be created and will only detect the mutex relation when supporting an effect which explicitly removes the value v.

Empirical results

In order to test our approach we use the benchmark set from IPC-3, the competition in which VHPOP competed, and test our results against those of the latest version of VHPOP using lifted actions. We have run two tests, first we show the results of using grounded actions with the stratification method and then we will detail the results of using the lifted representation. Our main focus is to reduce the search space, so the numbers denote the number of states visited before a solution was found, see Table 1. We allowed the planners to work for 10 minutes per problem instance, all problems unsolvable due to the imposed time constraint are marked with a ”-”.

Discussion and future work

Splitting the problem into subproblems

We note that the results for our approach using grounded actions does not yield very good results on all tested instances. Part of this can be attributed to the fact that, upon achieving each landmark layer, some causal links are broken and need to be achieved again, but this is not the main reason why we see these results. When we analyze the behavior of our planner and how it traverses through the landmark layers we

Problem	Landmarks and Grounded actions	Landmarks and Lifted actions	VHPOP
driverlog 01	66	79	47
driverlog 03	94	1672	1242
driverlog 06	48	-	210728
driverlog 07	-	59365	175
zeno 01	6	10	10
zeno 02	19723	2810	5439
zeno 03	2034	87	28
zeno 04	2482	-	1264
zeno 05	-	-	2572
zeno 06	-	5083	973
zeno 08	-	132451	-
depots 01	-	3880	26947
rovers 01	214	220	2075
rovers 02	72	123	96
rovers 03	632	518	21934
rovers 04	416	518	82
rovers 05	385	4610	1963
rovers 07	68728	-	126976
rovers 12	3876	-	6142
satellite 01	-	995	45
satellite 02	-	1131	1782
satellite 03	1454	5190	140
satellite 05	4313	1128	2829
satellite 07	-	1646	-
satellite 14	5407	-	-
satellite 16	16453	-	-
satellite 17	17887	-	-

Table 1: Plans visited

see a critical flaw occurring in every single instance where more resources are available to accomplish a task, e.g. multiple drivers and trucks are available to deliver the packages, we force the planner to make a decision. So, for example, when faced with a number of trucks to pick up a particular package, the planner has very little information to use to determine how hard it will be to get a particular truck to that location; the same applies for getting a driver into a truck. From the planner's point of view, it can make use of any available option from the landmarks and it is unable to discriminate between them, as the rest of the planning problem will only become apparent when advancing to the next layer. The planner is forced at higher layers (i.e. closer to the goal) to make a decision regarding the distribution of resources and how to make use of them, when there is too little information to guide its search and poor choices can lead to very poor performance.

Lifted representation vs grounded actions

In an attempt to overcome the above limitations we decided to plan with lifted actions, and associated every variable with a domain of values it can be assigned. During the planning process these domains are updated as the result of causal links and through separations in order to deal with threats. Using the SAS^+ representation we can detect mutex relations more quickly and if the planning problem has multiple resources to use these can be represented in the final plan (as in Figure 1).

In the above results we see that we are able to improve upon the grounded instance of splitting the problem up into subproblems, making effective use of the disjunctive landmarks. However, in bigger problem instances the benefit of having landmarks quickly dissipates and we are better off using grounded actions. One of the main problems is that although we create bindings from a variable to a set of objects, these bindings are often pruned within the same *landmark layer* which defeats the whole purpose of having these bindings in the first place. These issues usually stem from the fact that after splitting the problem into *landmark layers*, we disregard the orderings between the landmarks and bundle them together which confuses the planner.

A study of RealPlan (Srivastava 2000) shows that most planners, paradoxically, have more trouble finding a solution when given more resources. This is partly because most state-space planners ground all actions prior to planning, which can take up quite some time, but also because they tend to explore all possible actions from the current state. We have yet to show that our planner scales well as the problem size increases, but we believe that it has a lot of potential and that a lifted representation is key to get better scaling behaviour.

Future work

- When a SAS^+ variable has all its external dependencies satisfied and can make transitions within its DTG without changing external dependencies, we want to assume that that variable takes all those values at the same time. E.g. in the driverlog domain, a truck with a driver can visit all the locations a truck can drive towards. This, we hope, will prevent the planner from pruning the domains of the action variables prematurely.
- Develop heuristics which do not depend on grounded actions. We still use VHPOP's heuristics which require grounding of the problem, we want to forgo grounding all together.
- Take the ordering of landmarks into account during the planning process.
- Represent the bindings in a problem as a constraints problem, which allows for reasoning over the domains of variables.

Conclusions

In this paper we have merely laid the groundwork for partial order planning with lifted actions representation. However, there are still many points to improve upon as listed above. We think that implementing these will allow us to solve bigger problem instances and become more competitive with state-of-the-art grounded planners. The performance we achieve now is not competitive with the native lifted action implementation of VHPOP.

References

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.* 173(5-6):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)* 22:215–278.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)* 20:1–59.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning.

Porteous, J., and Sebastia, L. 2000. Extracting and ordering landmarks for planning. *J. Artif. Intell. Res. (JAIR)* 22:2004.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982. AAAI Press.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2006. Decomposition of planning problems. *AI Commun.* 19(1):49–81.

Srivastava, B. 2000. Realplan: Decoupling causal and resource reasoning in planning. In *In AAAI/IAAI*, 812–818. AAAI/MIT Press.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)* 20:405–430.