Improving relaxed-plan-based heuristics via simulated execution of relaxed-plans

Dunbo Cai[1], Minghao Yin[2], Jianan Wang[3]

[1]College of Computer Science and Technology, Jilin University, Jilin, China
[2]School of Computer, Northeast Normal University, Jilin, China
[3]Network Information Center, Northeast Normal University, Jilin, China
dunbocai@gmail.com, ymh@nenu.edu.cn, wangjn@nenu.edu.cn

Abstract

Relaxed plans, in the ignoring action's delete list relaxation mode, have been used as a good approximation of the goal-distances of search states. Recently, many researchers tried to improve relaxed plan based (RPB) heuristics by taking information from action's delete lists and got promising results. However, it is still a crucial point to keep the improved heuristics with low computational cost while discovering more information. We propose to improve RPB heuristics by gathering information from a simulated execution of relaxed plans. Based on the simulation, we combine the number of unsatisfied action's preconditions and top level goals as a penalty to a RPB heuristic. The penalty is sensitive to both the order and negative effects of actions in a relaxed plan and requests low overhead in computation in most cases in our experiment. Preliminary results show that our methods improve the FF's heuristic over several domains.

Introduction

Deriving heuristic functions from relaxed problems has got big success in domain independent satisficing planning (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert 2004). "Ignoring delete lists" is a popular way of making such relaxations. Based on the relaxation, the RPB heuristic in FF (h_{FF}) (Hoffmann and Nebel 2001) and the set-additive heuristic (Keyder and Geffner 2007) still show relative power to the recently proposed Causal Graph based heuristic h_{CG} (Helmert 2004) and the context-enhanced additive heuristic h_{cea} (Helmert and Geffner 2008) that take account of actions' delete effects to some extent. Particularly, h_{FF} is capable of capturing positive interactions among sub-goals, which can not be easily reasoned in the intrinsic *additive* heuristics, such as h_{CG} and h_{cea}. Hence, there is a lot of recent work towards improving RPB heuristics by taking into account the delete effects of actions in several ways, e.g., the learning based approach by Yoon et al. (2007); the TSP heuristic (Keyder and Geffner 2007) and the *Occlusion Penalties* (Baier 2007).

In this paper, we propose a simple and general way to consider negative interactions among sub-goals based on information from a relaxed plan. We simulate the execution of a relaxed plan while accounting for actions' delete effects, and combine the number of unsatisfied actions' preconditions and top-level goals as a simulated execution penalty (SEP) of the relaxed plan. The SEP and the original heuristic value together provide a new estimation of the goal distance of a state. Note that our idea can be applied to any kind of relaxed plans to take account of actions' delete effects, such as those generated by classical planning heuristic functions h_{FF} or the set-additive and the conformant planning heuristic function of Conformant-FF (Brafman and Hoffmann 2008). The underlying components include how to define the semantics of actions' application on (approximate) states and how to synthesize the penalty. In this paper we propose semantics for the simulation of classical relaxed plans (in section 3). As there are several ways to synthesize the penalty: qualitatively with thinking optimistically or quantitatively with thinking pessimistically, we'll provide empirical results on both. In addition, we'll show that our approach is sensitive to the order of actions in a relaxed plan and hence different from the approach in (Baier 2007). The simulation of relaxed plans could serve as a criterion for knowing how close a relaxed plan is to a real plan in terms of the actions and the order of actions.

We apply our approach on h_{FF}, and get two heuristic functions called $h_{PEO\text{-}FF}$ and $h_{PEP\text{-}FF}$. Experimental results on benchmarks domains show that $h_{PEP\text{-}FF}$ gains remarked improvement on h_{FF} on several domains.

The paper is organized as follows. The next section introduces backgrounds of planning. In section three we motivate our approach by an example, and then define the semantics for executing classical relaxed plans and the aggregation of penalties. Following that, we give the experimental results of our approach on several benchmark domains. We make some discussions and conclude at the end.

Background

A planning task is $\boldsymbol{T} = (\boldsymbol{F}, \boldsymbol{O}, \boldsymbol{I}, \boldsymbol{G})$, where $\boldsymbol{F}$ is the set of atoms involved, $\boldsymbol{O}$ is the set of actions, $\boldsymbol{I} \subseteq \boldsymbol{F}$ is the initial state, and $\boldsymbol{G} \subseteq \boldsymbol{F}$ is the goal conditions. We follow the representation of actions in (Hoffmann and Nebel 2001). An action $o \in \boldsymbol{O}$ is of the form ($pre(o)$, $add(o)$, $del(o)$), where $pre(o)$, $add(o)$ and $del(o)$ are subsets of $\boldsymbol{F}$ and denote preconditions, add list and delete list of o respectively. A state s is a subset of $\boldsymbol{F}$. An action o is *applicable* in s if $pre(o) \subseteq$ s. Here, we assume each action with cost 1. The planning problem on $\boldsymbol{T}$ is to find a sequence of actions that maps the initial state $\boldsymbol{I}$ into a goal state s_G that satisfies $s_G \subseteq \boldsymbol{G}$.

The *delete-relaxation* of $\boldsymbol{T} = (\boldsymbol{F}, \boldsymbol{O}, \boldsymbol{I}, \boldsymbol{G})$ is $\boldsymbol{T'} = (\boldsymbol{F}, \boldsymbol{O'}, \boldsymbol{I}, \boldsymbol{G})$ where $\boldsymbol{A'} = \{(pre(o), add(o), \varnothing) | (pre(o), add(o), del(o)) \in \boldsymbol{A}\}$. The cost of $\boldsymbol{T}$ is estimated by the length of a plan of $\boldsymbol{T'}$, which is computed by the relaxed GraphPlan in h_{FF} (Hoffmann and Nebel 2001) or by mathematical equations in the set-additive heuristic (Kedyer and Geffner 2007).

The RPB heuristics can capture the so-called positive interactions among goals: if there is an action o that adds two facts p and q and is applicable in a state s, then the cost of reaching p and q from s is estimated by the RPB heuristics as 1. In contrast, the additive heuristics may count o twice in the above example, which is not reasonable. However, as delete-lists of actions are ignored, RPB heuristics cannot reason about negative interactions among sub-goals. Take a simple example. There is an action o' that adds a goal fact p and deletes a goal fact q. If q was reached and o' is used to achieve p then q is destroyed and should be re-achieved with an additional cost. RPB heuristics will ignore the additional cost and hence misguide a search algorithm in some cases as discussed in (Helmert 2004). In the following section, we'll use the delete effects of actions in relaxed plans to take account of such additional cost, which can also uncover problem structures to some extent.

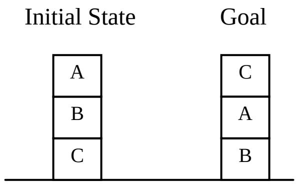

Figure 1: Blocksworld Example.

Enhancing RPB heuristics with penalties

We firstly illustrate our idea by the example in Fig. 1. In the planner FF, a relaxed plan for the state s is : <{unstack(A,B)},{unstack(B,C)},{pickup(C)},{stack(C,A) }>. Thus, the heuristic value for s is 4, which under-estimates the real optimal goal-distance that is 8. To improve the estimate, we'd like to know how much additional cost should be paid to extend the relaxed plan be a real plan. We could estimate the cost by a run of the relaxed plan with recording the number of unfulfilled preconditions (*flaws*) of actions in the relaxed plan. Table 1 lists the actions in order in the relaxed plan and also the global goals. Let's make a run of the relaxed plan and see what happens. The preconditions of the first action unstack(A,B) are all fulfilled in the state s. After executing unstack(A,B) with both its add list and delete list on s, we get an approximate state s' where the fact *handempty* is deleted. For the second action unstack(B,C), its precondition *handempty* is not fulfilled in s'. To fix the flaw, we need a sequence of actions that can apply after unstack(A,B) and adds *handempty* (in this case, putdown(A) can do the job). With the fix, we may execute unstack(B,C) on s' with adding the add list and deleting the delete list of unstack(B,C). Following this way, we may found: the third action pickup(C) has *handempty* unfulfilled; the forth action stack(C,A) has *clear(A)* unfulfilled and finally the global goal *on(A,B)* is not fulfilled. Totally, we need 3 additional actions to fix the flaws in actions' preconditions and 1 action for the goal *on(A,B)*. With these information, we propose to update the heuristic value of s be 4+(3+1)=8. We call the cost 3+1 as a penalty for the RPB heuristic.

We should note that getting the optimal estimation in the above example is a coincidence anyway. However, the simulated run of the relaxed plan in Table 1 really uncovers some negative effects of actions. Especially, our approach considers the availability of the *hand*, which is ignored in the relaxed problem.

unstack(A,B)	unstack(B,C)	pickup(C)	stack(C,A)	Global goals

pre:
clear(A)
on(A,B)
handempty
	pre:
	clear(B)
	on(B,C)
	handempty
		pre:
		clear(C)
		handempty
			pre:
			holding(C)
			clear(A)
				pre:
				on(A,B)
				on(C,A)

Table 1: Actions in a relaxed plan.

Simulated Execution of Relaxed-Plans

Given a planning task $\boldsymbol{T} = <\boldsymbol{F}, \boldsymbol{O}, s, \boldsymbol{G}>$ and a relaxed plan for s of the form $\pi = <a_0, \ldots, a_{n-1}>$. The simulated execution of π is a sequence of state $<s_0, s_1, \ldots, s_n>$, where $s_0 = s$, $s_i = s_{i-1} + pre(a_{i-1}) + add(a_{i-1}) - del(a_{i-1})$ for $i = 1..n$. Note that the simulated execution is different from real execution of a_{i-1} on s_{i-1}: $s_i = s_{i-1} + add(a_{i-1}) - del(a_{i-1})$, to reflect the existences of fixes. We assume that there are always some sequence of actions add the unfulfilled preconditions of a_{i-1} in s_{i-1}: $pre(a_{i-1}) - s_{i-1}$.

We assume a relaxed plan is a sequence of actions. The order of actions in the relaxed plan is set by the underlying algorithms constructing the relaxed plan. We'll explain this on h_{FF} later. Also note that, we may need an exponential number of actions to achieve one unfulfilled precondition of an action. So, we make further simplifications in the following section to aggregate penalties.

Aggregate Penalties

From the definition of the simulated execution, if a_i is not applicable on the state s_{i-1}, we assume there are some additional actions that can reach facts in $pre(a_{i-1}) - s_{i-1}$. If there are more than one unfulfilled preconditions, we propose two ways to estimate the number of actions that are needed. If thinking optimistically, we could use one action to achieve all the unfulfilled preconditions. If thinking pessimistically, we could use one action to achieve one such precondition only. According to these two options, we define the penalties of relaxed plans in the following two ways.

For convenience, we add a dummy action into $\boldsymbol{T} = <\boldsymbol{F}, \boldsymbol{O}, s, \boldsymbol{G}>$ to represent goals: a_G with preconditions $\boldsymbol{G}$, an add list $\{g\}$ and an empty delete list. The penalty of a relaxed plan $\pi = <a_0, \ldots, a_{n-1}>$ for $\boldsymbol{T}$ is:

$$P(s,\pi) = \sum_{i=0..n} ER(s_i, a_i) \quad (1)$$

where s_i is generated in the simulated execution of π and $a_n = a_G$, and

$$ER(s_i, a_i) = \begin{cases} 0 & if \ \ pre(a_i) \subseteq s_i \\ 1 & else \end{cases} \quad (2)$$

or

$$ER(s_i,a_i) = |pre(a_i) - s_i| \quad (3).$$

If be optimistic, we use equations (1) and (2); otherwise we use equations (1) and (3). However, thinking optimistically may seem not realistic in domains where a set of facts can not be reached at the same time by any action. Our future work would consider to use a sequence of actions to fix a flaw when aggregating penalties.

Penalty-enhanced RPB heuristics

Given a RPB heuristic function h, for a state s and a relaxed plan π defined by h for s, we define the penalty-enhanced version of h as:

$$h_{PE}(s) = h(s) + P(s, \pi) \quad (4).$$

The penalty-enhanced RPB heuristic uses both the solution length of a relaxed plan and the cost of fixing the flaws in our "simulated execution of the plan". Hence, it has the potential to be more informative. $P(s, \pi)$ can be calculated in time $O(n|\boldsymbol{F}|)$, where n is the number of actions in the relaxed plan and $\boldsymbol{F}$ is the set of atoms in the planning task.

Our approach can be applied to any kind of RPB heuristic that defines a relaxed plan while making estimations of states. And, it will be interesting to see what will happen when our approach applied to several RPB heuristics, as the penalty $P(s,\pi)$ depends on the order of actions in a relaxed plan. In the next section, we provide results on the well-known heuristic h_{FF}.

Experimental Evaluation

We applied our approach on FF, and tested nine benchmark domains. We implemented the two versions of our approach with the code of FF-2.3[1]: the optimistic version $h_{PEO\text{-}FF}(s) = h_{FF}(s) + P(s,\pi)$ with $P(s,\pi)$ defined by equations (1) and (2); the pessimistic version: $h_{PEP\text{-}FF}(s) = h_{FF}(s) + P(s,\pi)$ with $P(s,\pi)$ defined by equations (1) and (3). The search algorithms are Enforced Hill-Climbing (EHC) and Greedy Best-First (GBF), where we try EHC first and can GBF after EHC fails. The nine tested domains are: Assembly, Blocks (with 4 operators), Schedule, FreeCell, Depot, Rovers, Airport, Pips-notankage, and Trucks. All the instances in every domain are from the test suites of recent IPCs (1-5). The test was done on a computer running Linux with a 3.2GHz Intel Xeon CPU. The time limit is 300s and the memory limit is 512M. We report here the results of h_{FF}, $h_{PEO\text{-}FF}$ and $h_{PEP\text{-}FF}$, in terms of the failing rate, average node-expansions and average search-time over commonly solved instances.

[1] http://members.deri.at/~joergh/ff.html

The relaxed plans computed by FF following some strategies, such as "noop-first" and "actions with least difficulty first" (Hoffmann & Nebel 2001). These heuristics are proved to be useful. However, they are greedy to put actions at their earliest applicable time points in the relaxed problem. For example, given three locations A, B, C where A is connected to B and C, but B and C are not connected. If we want to traverse B and C from A to finish some logistic task, it is not reasonable to execute move(A,B) and move(A,C) both at the first time point. If we can order these actions in a more reasonable way (Keyder and Geffner 2007), then other actions, such as "loading packages", may be ordered accordingly in some way. This is one of our future working directions.

Before reporting the detailed data, we give a brief summary on the result. In the results, $h_{PEO\text{-}FF}$ and $h_{PEP\text{-}FF}$ show different performance on nearly a half of the testing domains. From the overall results, we see that $h_{PEP\text{-}FF}$ is much better than $h_{PEO\text{-}FF}$. Compared to h_{FF}, $h_{PEP\text{-}FF}$ and $h_{PEO\text{-}FF}$ have a slight increase in failing rate, but show remarked improvements in efficiency on 6 and 5 domains respectively.

	Number of unsolved Instances		
Domains	h_{FF}	$h_{PEO\text{-}FF}$	$h_{PEP\text{-}FF}$
Assembly (24)	0	0	0
Blocks (35)	4	3	0
Schedule (150)	17	19	26
FreeCell (20)	0	2	2
Depot (22)	2	3	3
Rovers (40)	4	0	0
Airport (34)	5	6	2
Pipes-no (34)	4	8	6
Trucks (30)	19	19	19
Total (389)	55	60	58

Table 2: Number of unsolved instances by domain.

Table 2 shows the failing rate of the tested heuristics. From the table, we can see that $h_{PEO\text{-}FF}$ and $h_{PEO\text{-}FF}$ h_{FF} fails a little more than h_{FF}. $h_{PEO\text{-}FF}$ is better than h_{FF} on Blocks and Rovers. $h_{PEP\text{-}FF}$ is better than h_{FF} on Blocks, Rovers and Airport. The three heuristic have difficulty in solving instances from the harder domains: Schedule (in IPC-2), Pips-notankage (in IPC-4) and Trucks (in IPC-5) within the given resource limits. We note that, in the Blocks and Rovers, h_{FF} fails on some large instances, which are all solved by $h_{PEP\text{-}FF}$. In other domains, the three heuristics often fails on large instances due to the limits on the resource. Our new heuristics also fails on some instances of moderate size that are solved by h_{FF}.

	Average Node-expansions				
Domains	h_{FF}	$h_{PEO\text{-}FF}$	$h_{PEO\text{-}FF}/h_{FF}$	$h_{PEP\text{-}FF}$	$h_{PEP\text{-}FF}/h_{FF}$
Assembly	204.25	188.83	0.92	188.54	0.92
Blocks	11209.70	705.94	0.06	225.90	0.02
Schedule	238.07	93.70	0.39	95.93	0.40
FreeCell	812.24	2621.59	3.23	651.24	0.80
Depot	12281.90	3293.56	0.27	3997.72	0.33
Rovers	3440.81	681.03	0.20	498.44	0.14
Airport	39464.80	5933.89	0.15	2748.81	0.07
Pipes-no	6696.09	10669.40	1.59	7974.64	1.19
Trucks	94835.70	219094.00	2.31	166678.00	1.76

Table 3: Average Node-expansions on commonly solved instances by domain.

	Average Search-time (s)				
Domains	h_{FF}	$h_{PEO\text{-}FF}$	$h_{FF}/h_{PEO\text{-}FF}$	$h_{PEP\text{-}FF}$	$h_{FF}/h_{PEP\text{-}FF}$
Assembly	0.19	0.15	1.27	0.15	1.27
Blocks	0.45	0.03	15.00	0.01	45.00
Schedule	0.28	0.09	3.11	0.09	3.11
FreeCell	1.40	6.15	0.23	0.86	1.63
Depot	4.58	2.65	1.73	2.54	1.80
Rovers	9.76	1.61	6.06	1.02	9.57
Airport	10.59	1.81	5.85	1.08	9.81
Pipes-no	2.49	4.65	0.54	4.03	0.62
Trucks	8.74	54.08	0.16	31.65	0.28

Table 4: Average Search-time on commonly solved instances by domain.

Table 3 shows the performance in terms of average node-expansions on "commonly solved instances" (those instances that are solved by the three heuristics). $h_{PEO\text{-}FF}$ shows big improvement on h_{FF} in Blocks, Schedule, Depot, Rovers, and Airport and a slight improvement on Assembly. Further, $h_{PEP\text{-}FF}$ has the same advantage and does better than h_{FF} on FreeCell. Compared to h_{FF}, the save in average node-expansions of our penalty-enhanced versions is more than 50% in Blocks, Schedule, Depot, Rovers, and Airport. In FreeCell, Pips-notankage and Trucks, $h_{PEO\text{-}FF}$ and $h_{PEP\text{-}FF}$ are worse than h_{FF}, which may indicate that the penalty computed by our approach is crude in domains with complicated problem structures.

Table 4 lists the average search-time of the tested heuristics. The table shows that the save in node-expansions makes $h_{PEO\text{-}FF}$ and $h_{PEP\text{-}FF}$ gain much improvement in search-time. In Blocks, Rovers, and Airport, $h_{PEP\text{-}FF}$ speeds up the search algorithm considerably, compared with h_{FF}. From table 3 and 4, we can conclude that the computation of penalties requires very low overhead in many cases.

Conclusions

We proposed a method to improve the RPB heuristics. The basic idea is based on that relaxed plans are usually useful guidance for constructing real plans. With actions' delete effects, we define the simulated execution of relaxed plans, and then aggregate penalties from the number of unfulfilled actions' preconditions and goals. The penalty serves as an estimate of the cost of extending a relaxed plan to a real plan. Preliminary results show that our approach improves the heuristic function of FF a lot on several domains, but also has difficulty to scale up in some domains with complicate problem structures.

The future work would be improving our approach by accounting some domain information in aggregating penalties and test our approach with the pure Greedy Best-First search algorithm to see the improvement in terms of accuracy. We'll also apply our approach to other RPB heuristics, such as the set-additive heuristic by Keyder and Geffner (2007) and the heuristic based on relaxed plan with low conflicts (Baier 2009).

Acknowledgements

We thank Hoffmann J. for sharing the code of FF-2.3. We also thank the reviewers for their very helpful comments.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. AI 129(1–2):5–33.

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Proc. ICAPS'04.

Keyder, E., and Geffner, H. 2007. Set-Additive and TSP heuristics for planning with action costs and soft goals. In Proc. ICAPS'07 workshop on Heuristics for Domain-Independent Planning.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Proc. ICAPS'08.

Yoon, S.; Fern A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In Proc. ICAPS'06.

Baier, J. A. 2007. Improving relaxed-plan-based heuristics. In Proc. ICAPS'07 workshop on Heuristics for Domain-Independent Planning.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In Proc. ICAPS'04.

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Proc. ICAPS'09.