

Planning with Action Prioritization and New Benchmarks for Classical Planning

Kamalesh Ghosh*, Pallab Dasgupta, and S. Ramesh

Indian Institute of Technology Kharagpur and
GM India Science Lab Bangalore
{kghosh,pallab}@cse.iitkgp.ernet.in, ramesh.s@gm.com

Abstract. We introduce a new class of planning problems in which there is a separate set of actions with higher priority than regular actions. We present new planning domains to show that problems of practical interest may easily fit in this framework. We argue that though this framework is quite succinctly encoded in classical planning itself, existing planners are disappointingly inept at solving them. To demonstrate this, we have built a wrapper tool for planners which uses ad-hoc techniques to give far better results. Therefore, we also propose our encoded domains as new challenges for general-purpose planners.

Keywords: Planning, benchmarks, action priority.

1 Introduction

Automated planning is an important area of research in Artificial Intelligence. Not only is it a very interesting research problem, but it also has real-life usage in areas like logistics scheduling and spacecraft commanding. It is widely recognized that automated planning is a very difficult problem by itself and in general, no planners can give any reasonable guarantee on performance. However, through successive improvements over the years, general purpose planners have become highly capable of solving a wide range of problems.

It is expected from general purpose planners that they are able to solve a wide range of problems without specific bias to any special characteristics of the problems. Based on this belief, we translated some special verification problems from the automotive control domain to planning. These verification problems are marked in their similarity with the planning problem itself, so translation is trivial and good results were expected. To keep it simple, we consider the original verification problem itself to be outside the scope of this paper and discuss our problem entirely from a planning perspective. We present some planning domains in this paper which are sufficiently endowed with the special characteristics of our original verification problems.

Basically, our typical planning problem has two independent sets of actions. As in classical planning, an initial state and planning goal is given and we need

* This work is partially supported by a research grant under the General Motors Collaborative Research Lab, IIT Kharagpur.

to find a sequence of actions that transform the initial state to a final state satisfying the goal. However, we have a special requirement which is to ensure during planning that we do not choose actions from the second set as long as any action from the first set is applicable. In other words, actions from the first set, which we term as the *control actions*, take an absolute priority over actions from the second set, termed as the *environment actions*. Additionally, the goal must be achieved in a state where no control actions are applicable.

As we show in this paper, this problem is easily reduced to classical planning so we can always encode and try to solve it with standard planners like FF[3] and LPG[4]. However, none of the planners that we have tried so far works well on our encoded instances. Consequently, we have formed the opinion that even domain-independent planners today have a bias towards solving some specific type of problems better than others. We believe that this bias has come because of an enthusiasm in the planning community with solving specific problems like the blocks world, logistics etc. In an attempt to help break off from this set pattern, we propose our encoded instances as new challenges for general purpose planners.

Our planning domains touch problems of practical interest in areas like power supply restoration [6] and functional model verification. Additionally, we argue that these problems are not as difficult to solve as the planners may make it appear because we are able to produce far better results through our own wrapper tool.¹

Thus, our main contributions in this work are that (i) we introduce a new class of planning problems for critical appreciation by the planning community and (ii) our encoded domains and tools give an opportunity to compare, investigate and improve on the existing capability of general-purpose planners.

The rest of this paper is structured as follows. We describe the basic framework of our class of planning problems in Section 2. In Section 3, we describe the encoding mechanism to support this framework in standard planners. In Section 4, we present a primary analysis of the reason that this encoding fails to work well in most planners. In Section 5 and 6, we present two domains which are modeled in our framework. Finally, in Section 7 we describe our wrapper tool and compare its results with the results achieved from planners through direct encoding.

2 Planning with Action Prioritization (PAP)

Our problem formulation is primarily based on propositional STRIPS planning [1]. A *state* is given by a set of propositions (i.e. ground atoms) belonging to a finite superset $\mathcal{P}$. The presence of a proposition indicates that its value is assigned to true in the state (and false otherwise). The precondition and postcondition of an action α , denoted by the symbols $\text{pre}(\alpha)$ and $\text{post}(\alpha)$, are finite sets of literals. The action set, $\mathcal{A}$, in our case is the union of two disjoint sets $\mathcal{A}_e$ and $\mathcal{A}_c$. We

¹ Our planning domains and tools can be downloaded from the link <http://www.facweb.iitkgp.ernet.in/~pallab/paplan.tar.gz>.

refer to $\mathcal{A}_e$ as the set of *environment actions* and $\mathcal{A}_c$ as the set of *control actions*. Thus an instance our problem, which we name as PAP, is actually a five-tuple $\langle \mathcal{P}, \mathcal{A}_e, \mathcal{A}_c, \mathcal{I}, \mathcal{G} \rangle$. As in propositional STRIPS planning, we are given an initial state $\mathcal{I}$ and a goal condition $\mathcal{G}$ (which is a set of literals) and we are required to find a sequence of actions which will transform the initial state to a state satisfying the goal condition. However, unlike in classical STRIPS planning, this sequence of actions (i.e. the *plan*) must have some special properties which we specify below.

A state is said to be a *control stable state* if and only if no control action is applicable in it. To keep things simple, we require that for each $\alpha \in \mathcal{A}_c$, there exists some $l \in \text{pre}(\alpha)$ such that $\bar{l} \in \text{post}(\alpha)$ (i.e., applying a control action makes it inapplicable). Now, considering the sequence of states that are visited as we apply the plan, we require the following:

1. Environment actions in the plan are applied only from control stable states.
2. The final state achieved by the plan is a control stable state.

There is one last issue that we must address before we can say that our problem is well defined. It is always possible that control actions are defined such that they form an inescapable loop in some states (meaning, we would keep applying control actions without any hope of reaching a control stable state). In this paper, we just assume that such situations are not allowed to occur.

Assumption 21. *In a valid PAP instance, a control stable state is always reachable from every reachable state.*

Unfortunately, this condition is not easy to verify. In practice, it may be possible to introduce stronger assumptions which are relatively easy to verify (e.g. disallowing any control loop, not just inescapable ones). As we intend to address PAP only in its most generalized form here, we consider this topic to be outside the scope of this paper.

2.1 Complexity of PAP

Finding whether a solution exists for a PAP instance is PSPACE-hard as an instance of the classical planning problem can be reduced to an instance of PAP by copying all the actions to the environment action set and leaving the control action set empty. On the other hand, we propose an algorithm in Section 3 which reduces PAP to classical STRIPS planning in polynomial time. This implies that the problem of checking whether a solution exists for a PAP instance is in PSPACE. Hence we deduce that PAP is a PSPACE-complete problem.

3 Compilation to Classical Planning

In this section, we show how PAP can be easily reduced to classical planning. The pseudo code of this reduction algorithm, named PAPPLAN, is given in



Fig. 1. Environment action E1 and control actions C1 and C2

Algorithm 1. We illustrate the working of this reduction algorithm on the set of actions shown in Figure 1.

The main emphasis of this reduction is on preventing the environment actions from being applied unless the state is a control stable state. In order to impose this restriction in a classical planning framework, we introduce one additional proposition $\text{dis}(C_i)$ for each control action C_i . The proposition $\text{dis}(C_i)$ is true only at states where C_i is not applicable. In order to express that an environment action applies only at a control stable state, we add all the $\text{dis}(C_i)$ propositions to its precondition (Algorithm 1, Line 6). For example, the precondition of E1 becomes $\{G1, \text{dis}(C1), \text{dis}(C2)\}$.

We say that an action α (environment or control) *triggers* a control action C_i if and only if the postcondition of the former contains one or more propositions that appear in the precondition of the latter. In our encoded instance, whenever we apply an action α which triggers a control action C_i , we set $\text{dis}(C_i)$ to false. The intent here is to prevent any environment action from being applied till we ensure that C_i is not applicable anymore. Therefore, we rewrite the postcondition of α to include $\neg\text{dis}(C_i)$ for each triggered C_i (Algorithm 1, Line 9). Using the shorthand $\langle \text{pre}(\alpha), \text{post}(\alpha) \rangle$ to express actions, the actions in our example in Figure 1 would finally become:

$$\begin{aligned}
 E1 &= \langle \{G1, \text{dis}(C1), \text{dis}(C2)\}, \{P1, G2, \neg\text{dis}(C1)\} \rangle \\
 C1 &= \langle \{P1\}, \{\neg P1, P2, \neg\text{dis}(C2)\} \rangle \\
 C2 &= \langle \{P2\}, \{\neg P2, \neg G1\} \rangle
 \end{aligned}$$

It is possible that C_i is not applicable after applying its triggering action α because some of the other propositions in its precondition are not true. Alternatively, C_i may later cease to be applicable after applying C_i or some other control actions. In both situations, we must be able to set $\text{dis}(C_i)$ back to true. For this, we introduce some new actions (for each C_i), each of which has the negation of a precondition of C_i as its precondition and $\text{dis}(C_i)$ as its postcondition (Algorithm 1, Line 12). For our particular example, they are:

$$\begin{aligned}
 M1 &= \langle \{\neg P1\}, \{\text{dis}(C1)\} \rangle \\
 M2 &= \langle \{\neg P2\}, \{\text{dis}(C2)\} \rangle
 \end{aligned}$$

The initial state and goal are modified suitably (Line 2, Line 3) and finally, we extract the plan by calling a classical planner (Lines 14-15). We do not argue the correctness of this reduction here due to space limitation, but believe that the reader will be readily convinced of it with a little effort.

Input: The propositions in the planning domain ($\mathcal{P}$), a set of environment actions ($\mathcal{A}_e$), a set of control actions ($\mathcal{A}_c$), the initial state ($\mathcal{I}$), and the goal state ($\mathcal{G}$).

Output: The generated plan ($plan$).

```

1  $\mathcal{P}' \leftarrow \mathcal{P} \cup \{\text{dis}(a_c) \mid a_c \in \mathcal{A}_c\}$ 
2  $\mathcal{I}' \leftarrow \mathcal{I} \cup \{\text{dis}(a_c) \mid a_c \text{ is not applicable in } \mathcal{I}\}$ 
3  $\mathcal{G}' \leftarrow \mathcal{G} \cup \{\text{dis}(a_c) \mid a_c \in \mathcal{A}_c\}$ 
4  $\mathcal{A}_d \leftarrow \emptyset$ 
5 foreach  $e \in \mathcal{A}_e$  do
6    $\text{pre}(e) \leftarrow \text{pre}(e) \cup \{\text{dis}(a_c) \mid a_c \in \mathcal{A}_c\}$ 
7 end
8 foreach  $a \in \mathcal{A}_e \cup \mathcal{A}_c$  and  $a_c \in \mathcal{A}_c$  do
9   if  $a$  is a trigger for  $a_c$  then
10     $\text{post}(a) \leftarrow \text{post}(a) \cup \{\neg \text{dis}(a_c)\}$       /* To mark  $a_c$  as enabled */
11  end
12 end
13 foreach  $a_c \in \mathcal{A}_c$  do
14   foreach  $\ell \in \text{pre}(a_c)$  do
15     $\mathcal{A}_d \leftarrow \mathcal{A}_d \cup \{\{\neg \ell\}, \{\text{dis}(a_c)\}\}$       /* To mark  $a_c$  as disabled */
16  end
17 end
18  $\mathcal{A}' \leftarrow \mathcal{A}_d \cup \mathcal{A}_e \cup \mathcal{A}_c$ 
19  $plan' \leftarrow \text{PLAN}(\mathcal{P}', \mathcal{A}', \mathcal{I}', \mathcal{G}')$ 
20  $plan \leftarrow \text{RemoveMarkerActions}(plan', \mathcal{A}_d)$ 
21 return  $plan$ 

```

Algorithm 1. PAPLAN

4 Our Encoding and Planning Graphs

Though all planners employ different heuristics and techniques, the most successful ones usually employ the planning graph [2] in some way or other. Since it is practically impossible for us to analyze the specific reasons of under performance for each and every planner, we focus on the planning graph and *mutex* relations to obtain a general picture. We assume that the reader is well aware of planning graphs and their construction method [5].

Let us again take the example in Figure 1 and assume that we are given the initial state $\mathcal{I} = \{G1\}$ and goal $\mathcal{G} = \{G1, G2\}$. It is obvious that this goal is not achievable because as soon as we apply E1, we have to next apply control actions C1 and C2 which would then destroy the goal G1. Remember that by definition we are required to achieve the goal in a control stable state.

The above example represents a very common phenomenon in planning problems featuring our prioritized action framework (as the reader may appreciate better in the next two sections). We will informally refer to this phenomenon as the *distant destruction* of goals. Basically, achieving of one literal using an action may necessitate destruction of others through successive triggering of control actions. A planner which is aware of such inevitable consequences would probably perform better than others.

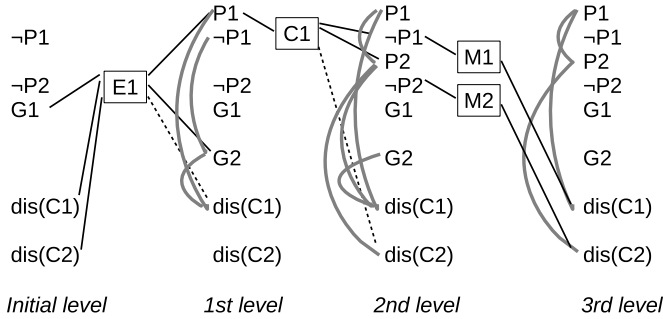


Fig. 2. Planning graph for encoded problem

Taking our encoded example, let us observe how the planning graph works. We partially depict the planning graph in Figure 2. Note that in order to prevent cluttering, we have only depicted those literals which are either goal literals or appear in some action's precondition. Persistence actions and some inconsequential actions have not been depicted either. A dashed line from the action to a literal indicates that the action destroys that literal.

At the first level, we are able to achieve G_2 but it is mutex with $\text{dis}(C_1)$ (as E_1 triggers C_1). This mutex carries on to the second level. However, as C_1 causes the deletion of the mutex between $\neg P_1$ and G_2 in the second level, we get no mutex between G_2 and $\text{dis}(C_1)$ in the third level (M_1 and persistence of G_2 are no longer mutex). So the planning graph predicts no difficulty in achieving the goal set $\mathcal{G}' = \{G_1, G_2, \text{dis}(C_1), \text{dis}(C_2)\}$.

Quite clearly, as the goal literals are achieved together pairwise in the intermediate levels, the planning graph loses all information relevant to distant destruction. We believe that this is an important drawback to the efficiency of any planner using planning graphs and mutex relations. We are presently working on techniques to address this and look forward to any future inputs from the planning community on this.

5 The Power Supply Restoration Planning Domain

The Power Supply Restoration (PSR) domain [6] is known to the planning community as it was featured in the fourth International Planning Competition (IPC-4). As our first working example of PAP, we present a model of this domain in our framework. What we like about our version of PSR is that we basically express the domain in plain STRIPS syntax, yet it remains just as succinct and easy to understand as the versions from IPC-4. The models of PSR from IPC-4 invariably make use of advanced syntactic constructs such as derived predicates or conditional effects that become unreadable when compiled to plain STRIPS.

The PSR domain basically models an electrical power distribution system. The model of a small power distribution system is illustrated in Figure 3. It

Next, we come to the **environment actions** which represent things that we can perform at will on the power distribution system, e.g. closing and opening of the switching devices. For example, we define the following two actions to represent the closing and opening of a switching device SD1, respectively:

$$\begin{aligned} & \langle \{ \neg \text{closed}(\text{SD1}), \neg \text{affected}(\text{CB1}), \neg \text{affected}(\text{CB2}) \}, \{ \text{closed}(\text{SD1}) \} \rangle \\ & \langle \{ \text{closed}(\text{SD1}), \neg \text{affected}(\text{CB1}), \neg \text{affected}(\text{CB2}) \}, \\ & \quad \{ \neg \text{closed}(\text{SD1}), \neg \text{fed}(\text{L1}), \dots, \neg \text{fed}(\text{L11}), \\ & \quad \neg \text{unsafe}(\text{SD1}, \text{side1}), \dots, \neg \text{unsafe}(\text{SD11}, \text{side2}), \\ & \quad \neg \text{unsafe}(\text{CB1}, \text{side1}), \dots, \neg \text{unsafe}(\text{CB2}, \text{side2}) \} \rangle \end{aligned}$$

Note that with each “open” action, we mark all the power lines as unfed and all the switching devices and circuit breakers as safe. This is because we intend to force a complete re-evaluation of the logic represented by the control actions in order to retain the correct values.

We have also have a “wait” action, quite similar to “open”, to represent the opening of circuit breakers specifically when they are affected (note that we are not allowed to perform normal opening and closing in that case). These actions mark the circuit breaker as unaffected in addition to opening it.

In Figure 3, a number of faulty lines are fed through the circuit breaker CB2, hence we must apply “wait(CB2)” before anything. Following this, lines L3 to L10 are left without power. To solve this example, we may open SD7 and SD11 first and then close SD3 to safely restore power to all non-faulty lines except L5. Finding out this sequence can be quite a difficult problem for large distribution systems. The reader is encouraged to satisfy himself/herself of the fact that our prioritized action planning would allow us to do such reasoning correctly.

6 The GRID Domain

We present here another representative domain for PAP. The system we visualize is a grid of *processors* serving job requests from one or more external *dispenser* units. A processor in our domain has four communication channels each of which can be connected to a neighbouring processor or dispenser unit. Examples of such interconnected systems are shown in Figure 4.

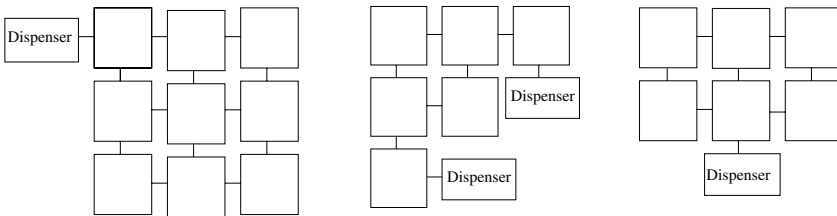


Fig. 4. Various configurations for a grid of processors

A dispenser unit creates new jobs and makes requests to one of its neighbouring processors to take them up. If the requested processor accepts, then it is instantly handed over the job. In the case of a reject, the dispenser waits till it eventually gets an accept signal from the requested processor. Though the dispenser receives an instantaneous response on its request, the internal allocation of the handling processor (or the collective decision to reject) happens through a non-trivial protocol which we clarify below.

Control Actions for GRID Domain. The processor allocation protocol dictates that each request be propagated through the grid in a distributed depth first manner. As soon as an available processor receives the request, it responds by giving an acceptance to the request. This acceptance is relayed back to the dispenser which then hands over the job description to be relayed back to the accepting processor. The communication channels allow the exchange of signals such as request, accept, reject, busy etc. and data such as job description between neighbours.

To make the problem more interesting, we assume that it is required that no processor handling a job should have more than two neighbours handling similar jobs. Also, no processor is allowed to handle two jobs simultaneously.

As a detailed description of the complete domain is beyond the scope of this paper, we just reproduce the English translation of some of the control actions below. Note that this is just a sample to illustrate the type of actions used and not a complete description:

- If any of the following three conditions hold, then mark self *unavailable*:
 1. The processor is *busy* handling a job.
 2. Three neighbours are busy with jobs.
 3. A busy neighbour has two busy neighbours.
- If none of the above conditions hold, mark self available.
- If a request for job processing is made by a neighbour and the current state does not have any outgoing requests (i.e., this is the first request to be processed in this round), then:
 1. If state of self is *unavailable*, make outgoing requests to all other neighbours besides the one making the request.
 2. If state of self is *available*, signal acceptance to the requesting neighbour.
- When transformed in to a state making outgoing requests to neighbours:
 1. If all requested neighbours signal reject or convey requests themselves (meaning that they cannot handle the job), then signal reject to requesting neighbours.
 2. If any requested neighbour indicates acceptance, then indicate acceptance to any requesting neighbour.
- If the processor (or dispenser) has a new job to dispense and a neighbour has indicated acceptance, hand over the new job to that neighbour and drop all outgoing requests and accepts.
- If the processor has received a new job description and is available itself, then get busy and drop outgoing accept signal.

- If state of self is such that requests have been made to three neighbours and the fourth neighbour (i.e. the original requester) has dropped its request, then drop all outgoing request and reject signals.

Environment Actions for GRID Domain. The main role of the environment in this domain is to submit and finish jobs. We maintain a total count of jobs at the system level (up to N , where N is the number of processors). The following environment actions are associated with each dispenser (there are N actions for each dispenser):

- If there are n jobs in the system and the dispenser is free, create a new job description in the dispenser and mark there to be $n + 1$ jobs in the system.

Likewise, the following environment actions are associated with each processor, representing the finishing of jobs (N actions for each processor):

- If there are $n + 1$ jobs in the system and the processor is busy, mark the processor to be not busy and the system to have n jobs.

Goals for Planning. Our planning goals typically represent some worst case scenario, very much like in verification problems:

1. Is it possible that the grid may not find a processor for the fourth job submitted?
2. Is it possible that the grid allows six jobs to run simultaneously?

7 Experiments

As the direct translation (Algorithm 1) fails to provide a satisfactory solution for our planning problems, we have created a wrapper tool which employs existing planners in an intelligent manner to solve these problems efficiently. This tool basically makes use of the same encoding as Algorithm 1, but in an incremental manner. The input language of this tool is the standard STRIPS subset of PDDL and we follow a simple naming convention to identify the control actions separately. We briefly explain the working of this wrapper tool here, and follow it with a comparison of the results achieved with it versus Algorithm 1.

There are basically four steps which our wrapper tool continuously iterates over till a solution is found:

Step 1, Compilation and planning. The original problem is translated partially and given to the classical planner, almost exactly as in Algorithm 1. By a partial translation we mean that many control actions do not have the $\text{dis}(C_i)$ propositions added or instrumented for them. In the very first iteration, the input instance is actually passed verbatim to the planner. Further iterations keep including more and more control action instrumentation.

Usually, the incomplete translation allows for easier plan generation, albeit in violation of the action priority requirements. For example, our illustrative problem in Section 4 will have a solution plan consisting of only the action E1 in the first iteration, which achieves both G1 and G2. This, clearly, is not an acceptable solution as C1 becomes applicable after applying E1.

Step 2, Rectification. After we generate a plan, we *rectify* it by patching it with applicable control actions. The **rectify** procedure basically scans the generated plan in order and inserts as many pending control actions as possible before each environment action (and at the end of the plan as well). For example, the rectification would give us E1.C1.C2 for our illustrative example. This is a very simple procedure that works on brute-force principle.

Step 3, Control Action Selection. If the planning goal is not met by a rectified plan, we then select some new actions to be restored to their special status as control actions (as mentioned in Step 1). We have two alternative policies to select new control actions. Our first variant, which we call the *Fast Convergence* policy, restores the status of all the control actions that needed to be inserted during rectification. The second variant, which we call the *Slow Convergence* policy, takes a more selective approach: only the control actions which were critical to the missing of the goal are restored.

Step 4, Macro Generation. We exploit the information from previous runs by introducing some environment-control interactions from last runs as *macro environment actions* for future runs. For example, the sequence E1.C1.C2 could be introduced as one macro action $\langle \{G1\}, \{\neg G1, G2, \neg P1, \neg P2\} \rangle$. By careful selection of macro environment actions, we are able to achieve great improvements in total solution time.

We compare the performance of direct compilation versus our wrapper tool. We use test cases of different sizes from the PSR and GRID domains and use three different planners FF, maxplan[7] and LPG to demonstrate the consistency of results. The scalability of different versions of the tool (that is, different combinations of optimization techniques and back-end planners) is illustrated by

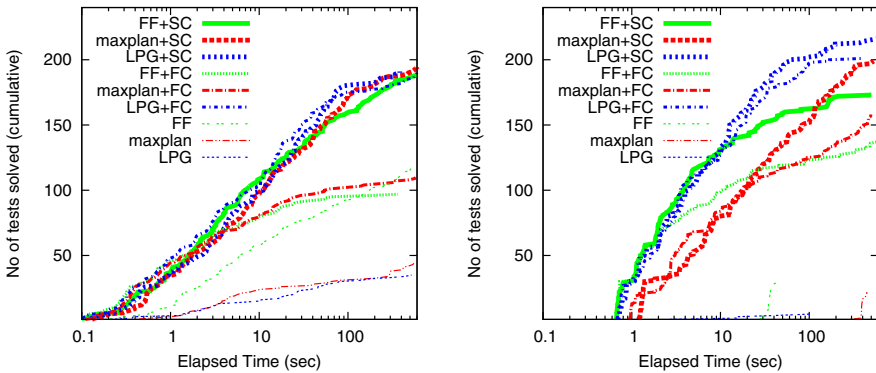


Fig. 5. No. of Tests Solved with Time for PSR and GRID domains

plotting the cumulative number of tests solved against elapsed time in Figure 5. We use a logarithmic scale for time as it gives a clearly visible and almost linear distribution of points. The curves for the versions of the tool using a direct compilation with a planner typically appear at the bottom, indicating that they solve fewer instances and take more time. The curves with the fast convergence policy (shown as +FC) appear in the middle while the best results are typically achieved by the slow convergence policy (+SC) for the same planners.

8 Conclusion

We strongly believe that the PAP formulation can prove useful for modeling many problems of practical interest. However, we realize that it may take much deliberation within the planning community to establish and treat it as a first class entity. For now, encoding it as classical planning seems to be the most legitimate solution but unfortunately, existing planners fail to satisfactorily solve it. We feel that the domains presented in this paper (i.e, their encodings) serve as a good challenge for general purpose planners as they may reveal some intriguing and commonly occurring features in difficult-to-solve planning problems which can be exploited by the planners.

References

1. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4), 189–208 (1971)
2. Blum, A., Furst, M.: Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2), 281–300 (1997)
3. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res (JAIR)* 14, 253–302 (2001)
4. Gerevini, A., Serina, I.: LPG: a planner based on local search for planning graphs with action costs. In: *Proc. ICAPS*, pp. 12–22 (2002)
5. Russell, S., Norvig, P.: *Artificial intelligence: A Modern Approach*, 2nd edn., ch. 11. Prentice Hall of India Private Limited, New Delhi (2006)
6. Thiébaux, S., Cordier, M.: Supply restoration in power distribution systems: A benchmark for planning under uncertainty. In: *Pre-Proc. 6th European Conference on Planning (ECP 2001)*, pp. 85–96 (2001)
7. Chen, Y., Zhao, X., Zhang, W.: Long-distance mutual exclusion for propositional planning. In: *Proc. IJCAI*, pp. 1840–1845 (2007)