Exploiting Opportunities Using Planning Graphs

Roman van der Krogt, Mathijs de Weerdt, and Cees Witteveen
Delft University of Technology
{r.p.j.vanderkrogt|m.m.deweerdt|c.witteveen}@ewi.tudelft.nl

Abstract

Opportunities arise in planning when changes in the environment make new propositions available to the planner. *Replanning* methods often focus only on the *negative* effects that changes in the environment have and therefore do not apply well. This paper introduces a method that is specifically focused on the opportunity problem. This method is based on the *potential graph* structure. This potential graph is basically an annotated version of the planning graph as used in many contemporary planners. We show how this potential graph can be used to discover the positive impact of opportunities. This information can then be used to improve upon the plan by exploiting such opportunities. The benefit of this method over standard planning techniques is two-fold: firstly, it can be used in an any-time algorithm. That is, when only little time is available, little improvements are quickly found. With more time available, greater improvements may be found. Secondly, this technique helps us focus on those parts of the plan that might be improved. Other parts are not changed, which greatly adds to the efficiency of the method. The applicability of the method is demonstrated by some initial experiments.

1 Introduction

Planning is an important ability for intelligent agents to accomplish the goals they have set for themselves, or to reach those that they have accepted from others. The planning problem involves choosing actions the agent has to carry out in order to move from the current state of the world to a state in which it has achieved its goals. The outcome of the planning phase is a *plan*: a (possibly partially ordered) list of actions that the agent has to take to achieve its goals. Nowadays, a large number of systems is available that can help solving the planning problem (see (Long and Fox, 2002) for an overview of state-of-the-art planning systems).

This paper does not deal with plan generation, however. Instead, we focus on the case in which a plan has already been constructed, and the planner is informed of changes in the environment that do not affect the plan, but

which might be exploited. We call such changes *opportunities*. There are two categories of methods that could be applied to this problem: planning and replanning methods. Planning methods can be used to generate a new plan from scratch. Although this approach works, it does not take into account the work that has already been put in constructing the current plan. It might well be that large parts of this current plan can be reused in the new plan. The other approach, replanning, is more focused on discovering and repairing things that go *wrong*. Since in this case nothing goes wrong, replanning methods are not very well suited either.

This paper tries to fill the gap that exists for such *opportunity problems*. The technique introduced by this paper to deal with such opportunities is called the *potential graph* method. A potential graph encodes all possible uses of an opportunity given some plan P and greatly facilitates in analyzing if and how such an opportunity should be used in improving this plan. The potential graph is, in fact, an annotated version of the planning graphs as introduced by Blum and Furst (1997). By effectively using the annotations, we can severely restrict the work that has to be done to extract improvements to the plan from these potential graphs. Moreover, by the incrementally extending the potential graph, we obtain an any-time algorithm, that quickly finds small improvements (if there are any) and, given more time, is able to extract bigger improvements.

The remainder of this paper is organised as follows. First we formally present the opportunity problem. Then we discuss the planning graph as used in Graphplan and extend these planning graphs by including annotations. Furthermore, we show how this annotated planning graph can be used to find positive effects of opportunities on a given plan. After that we present some initial experimental results. Finally, we conclude and give pointers to future work.

2 The Opportunity Problem

This section briefly recapitulates the classical planning and replanning problems in order to define our interpretation of some common terms. After that we introduce the Opportunity Problem, the central problem of this paper.

The Planning Problem In this paper we consider the STRIPS representation of classical planing problems. In this representation the state of the world at a given time is described by *propositions*, stating that a certain property is true. An example of such a set of propositions describing a state is $\{truck(A), at(A, \text{Amsterdam}), driver(A, \text{Tom})\}$, describing that a truck A is currently in Amsterdam and that its driver is Tom. *Actions* define transitions from one state to another. Each action a has preconditions $pre(a)$, and an effect list $eff(a)$, which is split in a addition list $add(a)$ and a deletion list $del(a)$. A *planning problem* $\Pi = (I, \mathcal{O}, G)$ in this representation consists of a set of initial propositions I and a set of goal propositions G, as well as a list of possible actions $\mathcal{O}$. The *solution* to a planning problem is a partially ordered set of actions

from $\mathcal{O}$, called a *plan,* P that can be executed from the initial propositions I and which achieves a state that satisfies the goals G.

The Replanning Problem Given a problem $\Pi = (I, \mathcal{O}, G)$ and a plan P that is a solution to Π, several things may happen due to a changing environment. For example, if a truck breaks down or a driver gets ill, some propositions are removed from the initial state. Customers having extra requests may cause the goal state to grow. In general, the problem may change to a problem $\Pi' = (I', \mathcal{O}', G')$ for which P may no longer be a valid solution, since P may use actions that are no longer present in $\mathcal{O}'$, may have preconditions that are no longer present in I', or attains only a subset of the goals G'. In these cases, the plan P has to be adapted to a plan P' that is a valid solution the Π'. Thus, a replanning problem consists of adapting a plan that has become faulty to a plan that is a valid to the new situation.

The Opportunity Problem The opportunity problem is a special case of the replanning problem as defined before. From a problem $\Pi = (I, \mathcal{O}, G)$ and a solution P , the environment changes in such a way that new propositions become available in the initial situation. Thus, the planner is faced with a new problem $\Pi' = (I', \mathcal{O}, G)$, where $I \subseteq I'$. The question now is: can we make use of these new propositions and improve the plan? Can we find a plan P' such that P' is a solution to Π', and P' is more optimal than P is, given the new situation?[1]

The solution we present to such opportunity problems is the *potential graph.* The potential graph is an annotated version of the Graphplan planning graph, in which special marks are used to label the proposition nodes. Using these labels, we can deduce which part of the planning graph has to be searched for improvements. First, the Graphplan planning graph is introduced, then we will discuss its extension to the potential graph.

3 The Graphplan Planning Graph

The Graphplan (Blum and Furst, 1997) planning graph is an acyclic directed leveled graph with two kinds of nodes. The levels alternate between *proposition levels* (containing proposition nodes) and *action levels* (containing action nodes). An action node for an operator o at level t (denoted by $[o]_t$) represents an (instantiated) operator o that can be planned at step t. A proposition node $[p]$ at level t corresponds to a proposition p that is *possibly* true at time t. Precondition-edges connect action nodes at level i to their preconditions at level i, addition-edges connect actions to their add-effects in level $i + 1$ and deletion-edges do the same for delete-effects at level $i + 1$.

The first level of the planning graph contains the propositions that are initially available (that is, the initial state I of a planning problem). Action level

[1] For this paper, we will assume that "more optimal" means "uses less actions".

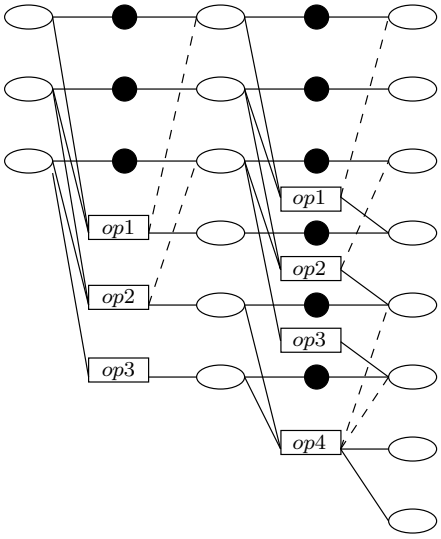

Figure 1: An example planning graph. Delete edges are represented by dashed lines. The other edges are either precondition or effect. No-ops are represented by black dots.

i ($i \geq 1$) contains all nodes corresponding to actions o for which the preconditions are present in proposition level i. Proposition level $i+1$ comprises all propositions that are the effect (either add- or delete-effect) of actions in action level i. Thus, the conditions imposed on a planning graph are much weaker than those imposed on valid plans: *all* actions that have their preconditions present are included in a layer. To record which of those actions can actually be used at the same time, the planning graph contains *mutual exclusion* (or simply *mutex*) relations. Two actions at a given level are mutex if no valid plan could possibly contain both at that same time. Similarly, propositions are mutex if they cannot be true at the same time. Two simple rules are used to find most of the mutual exclusion relations for actions.[2] The first declares two actions to be mutex when either of the actions deletes a precondition of the other. The second marks two actions that have some of their preconditions marked as mutually exclusive. Two propositions are marked as mutually exclusive when all ways of creating the first are exclusive of all ways that create the other.

Example 1 *An example planning graph is depicted in Figure 1. This planning graph starts with 3 initial propositions. Each of those can be propagated to the next level by means of a noop action (the black dots). Furthermore, there are three actions that can be instantiated using these propositions, each adding a new proposition. Actions 1 and 2 also delete one of their preconditions (depicted by dashed lines). Therefore, for example actions 2 and 3 are mutually exclusive:*

[2] Finding all mutex relations may take exponential time.

The expansion of a potential graph

The algorithm to expand a planning graph from i layers to layer $i+1$ proceeds as follows:

Algorithm 1 (EXPAND $(G, (I, P, \mathcal{O}, G), P_0)$)

Input: *A potential graph G with i layers, a context $(I, P, \mathcal{O}, G)$ and opportunities P_0*

Output: *The potential graph expanded with 1 level*

begin

1. **for each** *proposition node* $[p]_i$ **do**
   add a noop operator producing $[p]_{i+1}$
   if $[p]_i$ *is marked as dependent upon* P_0 **then**
   mark $[p]_{i+1}$ *similarly*
2. **for each** *operator* $o \in \wr$ *and each way of instantiating it to propositions available at level* i **do**
   add an operator node $[o]_{i+1}$ *and its add and delete effects.*
   if *any precondition* $[p']_i$ *to* $[o]_{i+1}$ *is marked as dependent upon* P_0 **then**
   mark $[p]_{i+1}$ *similarly*
3. **for each** *propostion* $[p]_{i+1}$ *created* **do**
   if $p \in \bigcup_{a \in P} pre(a) \cup add(a)$ **then**
   mark $[p]_{i+1}$ *as present in* P

end

action 2 deletes the precondition to action 3.

4 The Potential Graph

Now we come to the definition of the *potential graph*. A potential graph is a planning graph for a specific context (in particular the set of initial propositions and the plan P) in which some propositions are marked for special properties.

Definition 1 *The* potential graph *of a set of propositions P_0 given a plan P for a planning problem $(I, \mathcal{O}, G)$ is a planning graph with the propositions $P_0 \cup I$ in the first layer. From there, a normal planning graph is constructed (as described in the previous section), with one difference. The proposition nodes of the potential graph can be annotated with two different marks:* utilisation *marks and* presence *marks. The former mark all propositions that are produced with the help of P_0 (also said to be* dependent upon *P_0), the latter denote that a proposition node is equal to a proposition being used in the plan (i.e. a proposition node $[p]_i$ is marked with a presence node if $p \in \bigcup_{a \in P} pre(a) \cup add(a)$.*

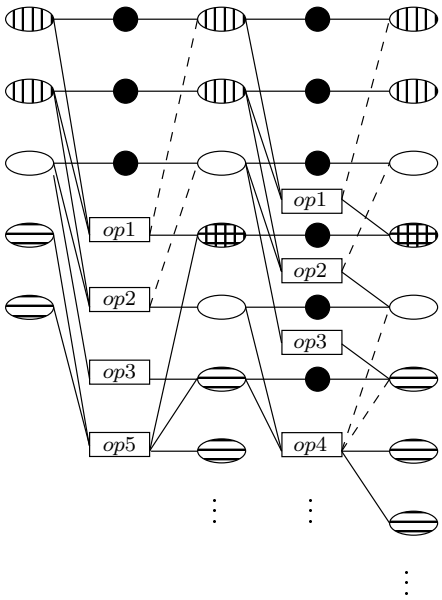

Figure 2: An example potential graph. Horizontal lines are used to mark propositions with utilisation marks, vertical ones denote presence marks.

The creation of a potential graph is analogous to the creation of a planning graph. From Theorem 1 of Blum and Furst (1997) we know that a planning graph can be constructed in polynomial time. Compared to a planning graph, the potential graph requires no extra size, except for the marks (a constant factor per proposition or action node). The only extra time needed is the time to decide for each proposition in the graph whether it is a precondition or effect of some action in the plan. This is clearly polynomial in the size of the plan. Therefore, the creation of a potential graph is polynomial in the length of the problem's description and the number of timesteps.

Example 2 *We refer to the planning graph of Example 1. Suppose that a plan is extracted from this graph that uses action 1 as its first action, possibly followed by some other actions.*

Now assume that two more propositions become available (i.e. 2 opportunities arise) and that the potential graph is constructed. Part of this graph is depicted in Figure 2. The two new propositions are horizontally shaded in the figure (marked as P_0). Vertically shaded are the preconditions to action 1 (marked with presence marks), and the effect of that action. Suppose that because of the new propositions, a new action 5 can be included in the graph. Its add-effects are marked with utilisation marks, since they are produced with the help of P_0. Note that the add-effect of action 1 is now marked twice: once for being dependent upon a proposition in P_0 and once for being present in the plan.

Proposition nodes $[p]_i$ that are marked twice are interesting: they are used by some action in the plan, and the potential graph shows that they can be obtained with the help of the new propositions. It is therefore possible that the subplan of the current plan P that satisfies this precondition p is larger than a subplan that uses some propositions in P_0. Thus, after every layer that is created, the propositions that are marked twice are examined to see whether they provide an opportunity to improve the plan. To do this, all plans P_X are extracted from the planning graph that may satisfy the interesting proposition. Extraction is done basically as it is done in Graphplan. That is, we use a level-by-level approach: given a set of goals G_t to attain at time t, we attempt to find a set of actions (including no-op actions) at time $t-1$ that have these goals as add-effect. We require that at least one of these actions has a precondition dependent upon P_0, to ensure that we are making use of the opportunity. The preconditions to those actions form the set of goals G_{t-1} to attain at time $t-1$. If we can find a plan that attains G_{t-1} at time $t-1$, the goals at time t can be achieved in t timesteps. Some of these goals G_{t-1} may be available in the plan already. In that case, we not only try to find actions producing these goals, but we also try to reuse these propositions from the plan. If all goals G_{t-1} are either present in the plan, or are propositions that are initially available, we have found a potential subplan to improve the present plan. This involves finding a set of actions that can be removed from P, where after the remainder of the plan and the extracted plan form a valid plan. Let a *used effect* of an action $a \in P$ be a proposition $p \in add(a)$ that is used to satisfy a precondition to some action $a' \in P$. (Partial order planners would say a causal link $a \xrightarrow{p} a'$ exists.) We can then find these actions in the following way.

First, we generate all combinations of satisfying preconditions from P_x by propositions from $\bigcup_{a \in P} add(a) \cup I$. For each of these combinations we can check (by simulating the execution of the plan) whether we can obtain a state in which all of the satisfying preconditions that we selected from P are true. From that state, we simulate the execution of the extracted plan P_x. Finally, by doing a backwards search from the goals, we can obtain the actions that can be executed from this state to obtain the goals. All actions that have not been executed during the simulation can be removed from the plan. The resulting plan P' is valid for obtaining G from I. If the number of actions that can be removed is greater than the number of actions contained in P_x then the plan P' that was created is a more optimal plan (in the number of actions used) than the plan P with which we started.

There are several benefits of the proposed solution over regarding the opportunity problem as a planning problem. First of all, in case an opportunity does not lead to an improvement in any way, we do not require an extensive search to find this information, as it will be clear from the marked propositions in the potential graph. Secondly, our method focuses the search on those parts of the plan where an improvement is attainable. Other parts of the plan are ignored, and thus require no computational resources. Thirdly, when using a planner, you either find no improvement or an almost optimal one. Using a potential

The (simplified) *Arithmetics* Domain

Below is a Strips-like representation of the operators in the toy domain. Note that this is not a particularly accurate description, since it omits the fact that the same number may occur multiple times in the same state. Adding a unique identifier to each number would be one way of solving this. Doing so, however, would make the operators needlessly complex.

```
operator addition::
  pre:  (number ?i)∧(number ?j)
  eff:  (not number ?i)∧(not number ?j)∧(number ?i+?j)


operator multiplication::
  pre:  (number ?i)∧(number ?j)
  eff:  (not number ?i)∧(not number ?j)∧(number ?i*?j)
```

graph, it is possible to find smaller improvements first and bigger improvements later on. This means we have an *any-time* algorithm.

5 Experiments

Initial experiments to investigate the merits of the potential graph have been carried out. As a proof of concept, we tried the method on a toy problem. This problem is the following: given a multiset I of (integer) numbers to start with and an integer number g to achieve, can we construct g from the numbers in I using only addition and multiplication operators?[3] A Strips-like encoding of this domain can be found in the sidebar. In this domain, we investigated the use of the potential graph method to optimize the current plan in the event of an additional number becoming available. Although this problem has limited practical use, it serves well to get an understanding of the process. The remainder of this section first quickly explains the planner that was built for this domain. After that, we show some results of applying the potential graph technique to opportunity problems.

To generate the initial plans to start our experiments with, a special-purpose planner was made. This plan uses simple domain-dependent heuristics (such as $3 \times 5 = 5 \times 3$) and backtracking to find a solution. This algorithm was created such that it quickly finds an initial solution and improves upon that. For example, within a second this planner finds an 11-step solution to the problem with $I = \{i, i+1 \,|\, 1 \leq i \leq 6\}$ and $G = \{331\}$. In about 2 seconds it is able to generate the 7-step plan of Figure 3. The planner takes about 19 seconds to prove that the optimal plan is a plan of length 4: $((3 \times 3) \times (6 \times 6)) + 7$.

[3] This problem was inspired by small puzzles given away for free in bags of candy.

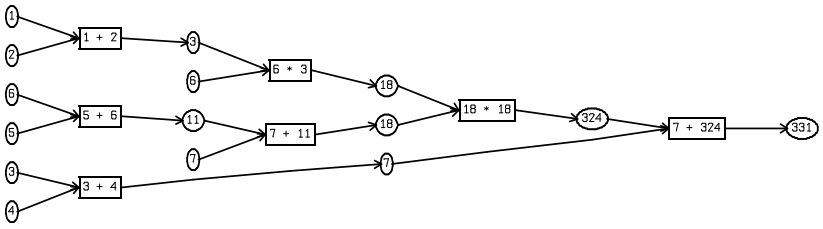

Figure 3: A plan with 7 steps to reach 331 from $\{i, i+1 | 1 \leq i \leq 6\}$. Only the numbers that are actually used ($\{1, 2, 3, 4, 5, 6, 6, 7\}$) are shown.

The experiments we conducted proceeded as follows: we generated n random input numbers with values in the range $[1, 5]$, and a goal in the range of $[15n, 20n]$. For these problems, we calculated an optimal plan, and then presented the system with a new available number of value 7. Before showing the resulting figures, we first make some general (empirically established) remarks. Firstly, running times are heavily dominated by the time it takes to create the potential graph (which takes about 95%), and more specifically by determining the mutual exclusion relations (approximately 80% of the running time). Secondly, because of the strict requirement on the plans that can possibly be extracted from the graph, only a very limited number of plans are actually extracted for further investigation. This is demonstrated by the fact that during the extraction phase, only a small portion of the potential graph is actually visited (about 11% of the proposition nodes). This shows that the extraction phase is really focused on a small part of the search space, where improvements can be found. This is further illustrated by the fact that only about 1.5% of the proposition nodes are marked with both presence and utilisation marks (which makes them potential candidates). Finally, note that we did not implement any of the existing known optimizations (such as (Long and Fox, 1999)) to Graphplan.

Figure 4 shows the running times of the potential graph technique. The figure depicts average runtimes for problems with 5 to 10 initial numbers, with one extra number (7) available as an opportunity, as described above. For each problem, two runtimes were obtained: *(i)* one using the standard potential graph creation algorithm as described above, and *(ii)* one in which the expansion is a two phase process in which first the marked propositions are expanded and searched for a solution (this implies that on the last level, only the marked propositions are expanded).

As one can see, the increase in time is almost linear in the number of initial numbers. There is a sharp bend in the figure between 7 and 8 inputs, this is caused by the fact that with 8 numbers, almost all of the potential graphs of this size require an extra level to be expanded (as the goals are set to higher values with more inputs). We can also observe that by using a two phase expansion strategy, we can cut down the running times almost in half.

Of course, we are also interested in a comparison with general purpose planners that are faced with the same task. In theory, planners supporting PDDL

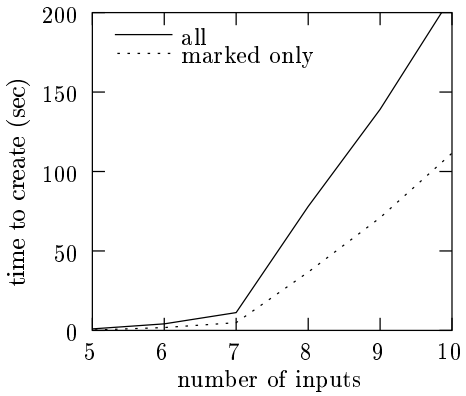

Figure 4: Experimental results

2.1 with numerical expressions (such as Metric-FF (Hoffmann, 2003), and LPG (Gerevini et al., 2003)) should support this domain. In practice however, we ran into some problems (e.g. Metric-FF does not support multiplication and LPG is not able to deal with goals on fluents). Comparison with the special-purpose planner built for this domain yielded no interesting results either. The special-purpose planner is able to deal with this domain so well, that the potential graph is no match for it. This comes as no surprise since domain-independent techniques cannot exploit the knowledge that domain-dependent planners can use.

6 Conclusions and Future Work

We started this paper by arguing that most agents cannot do without a means to deal with a dynamic environment, which changes due to causes outside of their control. We also noted that this problem is mostly tackled from one point of view: the plan repair view. This point of view focuses on how such unexpected events may negatively influence the plan, and how these negative effects can by countered by repairing the plan. The other point of view is often neglected: if an event does not have a negative effect on the plan, can the agent take profit from it, and improve its plan? This paper coined the term *opportunity* for such possibly positive events. To be able to deal with such opportunities, we introduced the *potential graph*, an annotated planning graph that can be used to tell what positive effects may be achieved due to this opportunity. The annotations are used to efficiently search through such graphs for improvements to the current plan.

We showed that a potential graph can be constructed in polynomial time (just like a planning graph) and also in a similar way. This also holds for the extraction phase: except for some small details this proceeds as the Graphplan extraction phase. However, whereas in Graphplan a single extracted plan is

enough, there may be multiple plans extracted from the potential graph. We showed how we can determine whether an extracted plan leads to an improvement to the plan.

Finally, we reported preliminary experimental results of this method. Although the presented results are intended as a proof of concept, they do show the validity of this technique. The experiments confirmed that the search for improvements is focused on that part of the search space where improvements can be found. They also showed that because of this efficient extraction, the generation of the potential graph (and more specifically the detection of mutual exclusions) plays a heavy role in the overall results.

The work described here is related to work done on *replanning* and *plan repair*, in the sense that we are also dealing with external events. Other work on such problems was done by Hammond (1990) and Drabble et al. (1997), for example. These methods investigate the impact of failing actions on the plan. They can be applied when an action fails (i.e. certain propositions are asserted that should not be, or vice versa) and the effects are required by other actions in the plan. In this case, they try to find actions to include in the plan to repair it. However, as we indicated before, such methods are not applicable when they are given (possibly) positive effects. This also holds for the work done by Gerevini and Serina (2000). Their work is of particular interest, because they too use planning graphs during replanning. (When a replanning problem is encountered, the associated planning graph is built and compared to the original plan to to find the implications of failures.) Once again, these systems all have a plan repair point of view and are not applicable to the problem that we try to tackle.

Further work obviously includes more general experiments using other domains, and hopefully comparison with a general purpose planner. Since the experiments show that the expansion phase is a major factor of the method, improvements to the potential graph generation should also be investigated. Of particular interest is whether heuristics can be developed that make it possible to cut down on the expansion of the potential graph. Another interesting research area would be to apply this technique in a multi-agent environment. Here, opportunities arise from the plans of other agents. As an example of this, the potential graph structure could be embedded in a *plan merging* algorithm (such as presented in Tonino et al. (2002)). These methods try to optimize plans of different agents in a multi-agent environment. They do this by looking for side-effects of each others plans: if the plans of agents A and B both include actions to reach some proposition p, than one of those plans can be optimized, by using the effect of the other plan. Usually, these methods only look at the *direct* applicability of each others effects. By using the potential graph method, it would be possible to not just merge plans by looking at what common effects are produced, but also by investigating further implications of each others side effects. A plan merging technique could also present a better way to solve the problem of determining whether some part of the plan can be replaced by the extracted plan from the potential graph.

Acknowledgements

Roman van der Krogt is supported by the Freight Transport Automation and Multi-Modality (FTAM) program of the TRAIL research school for Transport, Infrastructure and Logistics. Mathijs de Weerdt is supported by the Seamless Multi-Modal Mobility (SMM) program of the same research school. We thank the anonymous referees for their comments.

References

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300.

Drabble, B., Dalton, J., and Tate, A. (1997). Repairing plans on the fly. In *Proc. of the NASA Workshop on Planning and Scheduling for Space, Oxnard CA, USA.*.

Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of AI Research*. Accepted for the special issue on the 3rd International Planning Competition.

Gerevini, A. and Serina, I. (2000). Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pages 112–121.

Hammond, K. J. (1990). Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228.

Hoffmann, J. (2003). The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of AI Research*. Accepted for the special issue on the 3rd International Planning Competition.

Long, D. and Fox, M. (1999). Efficient implementation of the planning graph in stan. *Journal of AI Research*, 10:87–115.

Long, D. and Fox, M. (2002). International planning competition. http://www.dur.ac.uk/d.p.long/competition.html.

Tonino, J., Bos, A., de Weerdt, M. M., and Witteveen, C. (2002). Plan coordination by revision in collective agent-based systems. *Artificial Intelligence*, 142(2):121–145.