Replanning in a Resource-Based Framework

Roman van der Krogt, André Bos, and Cees Witteveen

Delft University of Technology
{R.P.J.vanderKrogt,A.Bos,C.Witteveen}@ITS.TUDelft.NL

Abstract. An important aspect of agents is how they construct a plan to reach their goals. Since most agents live in a dynamic environment, they also will often be confronted with situations in which the plans they constructed to reach their goals are no longer feasible. In such situations, agents have to change their plan to deal with the new environment. In this paper we describe such a replanning process using a computational framework, consisting of *resources* and *actions* to represent the planned activities of an agent.

1 Introduction

Often agents have to achieve a number of goals without a predefined way to accomplish them. Therefore, they have to make a plan that consists of a number of actions that, starting with the current situation, brings the agents to a desired state in which they have accomplished their goals. To assist an agent in such a planning task, a number of planning systems exists, e.g. Talplanner [4], Blackbox [9], FF [7] and many others. Most of such planning systems assume that the planning agent is the *sole cause of change* and that the actions have *deterministic effects*.[1] While these assumptions are necessary (but not sufficient) to make the planning process feasible, they usually no longer hold if an agent starts executing its plan in the real world. For instance, in a multi-agent environment, the agent will not likely be the only source of interactions with the world, violating the sole cause of change assumption. Moreover, in these situations, agents have to cope with unpredictable situations and actions. Hence, they are forced to adapt their original plan by *replanning* their actions.

A simple method of replanning is to give up the old plan and use a standard planning tool (such as mentioned before) to construct a new plan. This is, however, a rather inefficient approach: It will waste all the effort an agent has put in optimizing its current plan. Moreover, it might also violate agreements that have been made with other agents. Therefore, we propose a specialized plan revision method to adapt the current plan to a plan which takes into account the new situation. Unlike the approaches in [5] and [6] we do not need additional information gathered during the planning process itself, such as the reasons for adding certain steps, but we require the presence of a library of *plan schemes*, i.e. general plans that define the services an agent can provide. This library may

[1] See [14] for a discussion of planning systems and their assumptions.

V. Mařík et al. (Eds.): MASA 2001, LNAI 2322, pp. 148–158, 2002.

contain a set of possible (pre-compiled) plan repairs, but can also come from other agents that "advertise" their services. Each such scheme can be used to adjust the plan by either *adding* a subplan to it or *removing* a subplan from it.

The replanning method discussed in this paper attempts to find a combination of such plan adjustments that transforms the current plan into a plan that can be used in the new state of the world. The paper is organized as follows: We first present the Actions and Resource Planning Formalism (ARPF) that is used to represent an agent's plan. We introduce two basic operations to adapt such plans to a new situation, and show a replanning algorithm that uses these operations. After giving some initial experimental results, we conclude by comparing our work to that of others and by sketching the future work.

2 The Action and Resource Planning Formalism

This section gives an overview of the framework that we will use to model the plan of an agent. This formalism is described in detail in [3]. We model a process, like a production or transportation, by an *action*. An application of an action consumes a set of (input) *resources* and produces a disjunct set of (output) resources. These actions can be combined to reach certain goals. Such a combination is called a *plan*.

Resources. Each resource is identified by its *type* (a predicate symbol) and the set of *values* for its *attributes*. For example, $truck(5 : id, \text{StLouis} : loc, 15.^{00} : time)$ is the resource that describes truck number 5 being in St. Louis at $15.^{00}$.

A *resource scheme rs* is used to specify a set of resources sharing some attributes. Instead of ground values for attributes, a resource scheme may also contain variables specifying a set of values for an attribute.[2] For example, the resource scheme $truck(?i : id, \text{StLouis} : loc, ?t : time)$ refers to the set of all trucks that are in St. Louis at some point in time.

A resource r is an *instance* of a resource scheme rs, if there exists a substitution θ of variables to values such that $rs\theta = r$. Similarly, a set of resources R satisfies a set of resource schemes Rs, denoted by $R \models_\theta Rs$, if there is a *resource-identity preserving substitution* θ such that $Rs\theta \subseteq R$.[3] Finally, we have an *extended resource scheme*, which is a tuple $\langle Rs, C\rangle$, where Rs is a set of resource schemes and C is a set of constraints that may restrict the possible resources in the extended resource scheme Rs . For example,

$$\langle \{truck(?i_1 : id, \text{StLouis} : loc, ?t_1 : time), load(?i_2 : id, \text{StLouis} : loc, ?t_2 : time)\}, \{t_1 = t_2\}\rangle$$

denotes a truck and load in St. Louis at the same time.

[2] To distinguish between ground values and variables, a '?' is placed in front of a variable name, e.g., $?x$ denotes the variable x.

[3] A substitution θ is resource-identity preserving w.r.t. a set Rs of resource schemes if $\forall rs_1, rs_2 \in Rs \cdot rs_1 \neq rs_2 \rightarrow rs_1\theta \neq rs_2\theta$.

A resource set R *satisfies* an extended resource scheme $\langle Rs, C\rangle$, denoted by $R \models \langle Rs, C\rangle$ if for some resource-identity preserving substitution θ, $Rs\theta \subseteq R$ and $\models C\theta$, i.e. using θ all ground instances of constraints are valid. If the substitution θ has to be mentioned explicitly, we will also use $R \models_\theta \langle Rs, C\rangle$.

Actions. An action is a rule of the form $a : Rs_2 \leftarrow \langle Rs_1, C\rangle$. Here, a is the name of the action, Rs_2 is the set of resource schemes produced by a, Rs_1 is the set of resources schemes consumed by a and C the set constraints on Rs_1. For example,

$$\begin{aligned} \mathit{driveStL} : \{\mathit{truck}(?i : \mathit{id}, \mathrm{StLouis} : \mathit{loc}, ?t + d(?l, \mathrm{StLouis}) : \mathit{time})\} \leftarrow \\ \langle \{\mathit{truck}(?i : \mathit{id}, ?l : \mathit{loc}, ?t : \mathit{time})\}, \{?t > 7.^{00}, ?l \neq \mathrm{StLouis}\}\rangle \end{aligned}$$

is an action to drive a truck to St. Louis. From a truck at some location $l \neq$ StLouis at time $t > 7.^{00}$, it is possible to "produce" a truck in St. Louis at time $t + d(?l, \mathrm{StLouis})$. The set of input resources is denoted by $in(s) = Rs_1$, the output resources are denoted by $out(a) = Rs_2$. The application of an action transforms a set of resources R_1 into a set of resources R_2. Such an application is specified by a substitution θ changing each occurrence of a resource scheme in $in(a)$ and $out(a)$ to an occurrence of a fully specified resource. Let $a : Rs_2 \leftarrow \langle Rs_1, C\rangle$ be an action and let R_1 and R_2 be sets of resources. We say that R_2 can *immediately be produced from* R_1 *using* a, abbreviated by $R_1 \vdash_a R_2$, if there is a resource-identity preserving substitution θ such that (i) $R_1 \models_\theta Rs_1$, (ii) $\models_\theta C$ and (iii) $R_2 = (R_1 - Rs_1\theta) \cup Rs_2\theta$, i.e. all resources from $Rs_1\theta$ are removed from R_1 and the resources $Rs_2\theta$ are added. Generalising this production relation to a set of actions A, $R_1 \vdash_A R_2$ is said to hold iff there is some $a \in A$ such that $R_1 \vdash_a R_2$ holds. We will use $\vdash_A^*$ to denote the reflexive-transitive closure of $\vdash_A$.

Goals and Plans. A *goal* of an agent is a description of a resource the agent wishes to obtain. Therefore, goals will be described in terms of resource schemes. A goal g is a resource scheme. A *goal scheme* is an extended resource scheme $Gs = \langle G, C\rangle$, where G is a set of goals and C are constraints on G.

Let $\mathcal{R}$ be the set of resources and A the set of actions. Suppose that $R \vdash_A^* R'$, i.e., there exists some partially ordered set (poset) of instances of actions in A such that R' can be produced from R. This poset of instances of actions together with the ground instances of resources associated is called a *plan* P. Such a plan is a plan for a goal scheme Gs if, starting from the initial resources R, a set of R' resources is produced that satisfy Gs.

To represent a plan explicitly[4] , we use a (bi-partite) Directed Acyclic Graph $P = \langle N_\mathcal{R} \cup N_A, E\rangle$, where $N_\mathcal{R}$ is a set of *resource nodes* n_r with $r \in \mathcal{R}$, N_A a set of *action nodes* n_a where $a \in A$, and $E \subseteq (N_\mathcal{R} \times N_A) \cup (N_A \times N_\mathcal{R})$ is the set of arcs. For $n_r \in N_\mathcal{R}$ and $n_a \in N_A$, $(n_r, n_a) \in A$ means that resource r is used by an application of action a, and $(n_a, n_r) \in A$ means that resource r is

[4] Slightly abusing language, in the sequel we will use plans and plan graphs interchangeably.

produced by an application of a. A plan P consumes its set of *input* resources of P, denoted by $in(P)$, and produces its set of *output* resources, $out(P)$. Output resources that are not used to satisfy a goal are called *free resources*, denoted by $free(P, Gs)$, or just $free(P)$ if it is clear which goals Gs are to be satisfied. A plan P *realizes* a goal scheme Gs *using* a set of initial resources R and a substitution θ, denoted by $R \models_P Gs\theta$, if (i) $R \supseteq in(P)$ and (ii) $out(P) \cup (R - in(P)\theta) \models_\theta Gs$. If $R \models_P Gs\theta$, the triple (R, P, Gs) is called *adequate*.

Given a plan $P = (N_A \cup N_\mathcal{R}, E)$ and a subset $N'_A \subseteq N_A$, the subplan P' *generated by* N'_A is the subgraph P' of P generated by the set of nodes $N' = N'_A \cup \bigcup_{a \in N'_A} out(a) \cup in(a)$.[5]

Plan schemes are used to denote the services an agent can provide, i.e. ways of organizing actions to obtain goals, specifying the input resources needed to obtain them by providing (extended) resource schemes. Hence, plan schemes contain resource schemes to label the resource nodes. They can be simply defined as follows: If Ps is a plan scheme, θ a ground substitution that assigns a value to every variable occurring in Ps and every ground instance of a constraint occurring in Ps is valid, $Ps\theta$ is a plan. Plans, therefore, also can be considered as special cases of plan schemes.

3 Plan Repair and Plan Operators

Suppose that an agent is able to achieve a goal scheme Gs using a plan P with initial resources R, i.e., the triple (R, P, Gs) is adequate. Since the agents operate in a dynamic environment, initial resources as well as the availability of actions and the goals to be achieved might change. So, after some time, the agent might discover that instead of the adequate triple (R, P, Gs), the actual set of available resources is R', the realizable part of his original plan is P', the actual goal scheme is Gs' and the triple (R', P', Gs') is no longer adequate.

Hence, his current plan P' has to be adapted. We thus concentrate on the following *replanning problem*:

> Given an adequate triple (R, P, Gs), an actual set of resources R', a realizable part P' of P and a goal scheme Gs', find a plan P'' such that (R', P'', Gs') is an adequate triple, i.e., there exists some substitution θ such that $R' \models_{P''} Gs'\theta$.

We will solve this problem by *changing* the plan P' to P''. Note that P' might fail for exactly one or both of the following reasons: (i) $out(P') \not\models Gs'$ i.e., P' is not able to satisfy the current goals; (ii) $in(P') - R' \neq \emptyset$, i.e., P' lacks some resources to satisfy the goals.

Note that both cases (and their combination) can be specified as a triple (R', P', Ers) where Ers is an extended resource scheme, specifying the (set of) resources needed in addition to P' and R' to satisfy all goals.[6]

[5] The subgraph of $G = (N, E)$ generated by a subset $N' \subseteq N$ equals $G' = (N', (N' \times N') \cap E)$.

[6] That is, there exists a substitution θ such that $(R' \cup Ers\theta, P', Gs')$ is adequate.

In order to obtain such a set of *missing resources* satisfying *Ers*, we might try to *add* a plan P_{add} to P' having *Ers* as its goal scheme such that the resulting plan can use R' to satisfy Gs'. Addition alone however might not suffice: sometimes we first have to *delete* some part of the existing plan and then add another plan to achieve the additional goals.[7]

We now describe these plan addition and deletion operators and then show that their combination is sufficient to guarantee a successful change whenever there exists a solution to the replanning problem.

Addition. The addition operator $\oplus$ adds two plans P and P' to form a larger plan $P'' = P \oplus P'$. Addition of plans, like deletion, however, is defined if they are *compatible* w.r.t. their resources and actions. Therefore, we will first define this compatibility relation:

Definition 1. *Let $P = (N = N_{\mathcal{R}} \cup N_A, E)$ and $P' = (N' = N'_{\mathcal{R}} \cup N'_A, E')$ be plans over a set of resources $\mathcal{R}$ and actions A. Then P and P' are said to be* compatible *if the following conditions hold:*

- *for every node $n \in N_{\mathcal{R}} \cap N'_{\mathcal{R}}$, ($(n,a) \in E \wedge (n,a') \in E'$ or $(a,n) \in E \wedge (a',n) \in E'$) implies $a = a'$; That is, if P and P' have common resources, then these resources are neither produced nor consumed by different actions.*
- *for every action $a \in N_A \cap N'_A$, $out_P(a) = out_{P'}(a)$ and $in_P(a) = in_{P'}(a)$; That is, if P and P have actions in common, then these actions consume and produce the same sets of resources.*
- *$(N \cup N', E \cup E')$ is an acyclic graph.*

Together these conditions guarantee that $P'' = (N \cup N', E \cup E')$ is a plan. Hence, for two compatible plans $P = (N, E)$ and $P' = (N', E')$, $P \oplus P'$ is simply defined as the plan $(N \cup N', E \cup E')$.

Example 1. Figure 1 shows an example use of the $\oplus$-operator. The different letters denote different resources, the numbered boxes represent actions. Action 2 is common to both plans, as are the resources C, D, E and F. The resulting plan has one new input resource, H, but does no longer require the resource D.

Deletion. Deleting a plan P' from a plan P is denoted as $P \ominus P'$, and is done by (i) removing the *common action part* of P and P' from P and generating the resulting plan from the remaining actions. So, for two compatible plans $P = (N_A \cup N_R, E)$ and $P' = (N'_A \cup N'_R, E')$, $P'' = P \ominus P'$ is defined as the subplan of $P \oplus P'$ generated by the set of remaining actions $N''_A = (N_A \cup N'_A) \setminus (N_A \cap N'_A)$.

Example 2. Referring to Fig. 1, if we would not add, but subtract the plan, the result would be the plan with actions 1 and 3, and resources A, B, C, E, G.

The following result can be easily proven:

[7] For example, if there are not enough input resources for the combined plan.

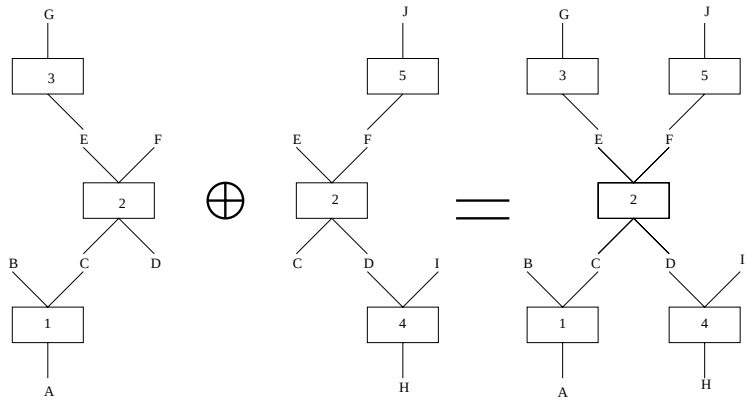

Fig. 1. Example use of the $\oplus$-operator. Here, a plan that produces resources B, F and G is shown on the left and we add to it and instantiated plan scheme that produces E, I and J.

Proposition 1. *Given an inadequate triple* (R, P, Gs) *and a plan* P' *be such that* (R, P', Gs) *is adequate. Then there always exist plans* P_1 *and* P_2 *such that* $P" = (P \ominus P_1) \oplus P_2)$.

Note that a sequence of a deletion followed by an addition also can be described by taking a replacement operator $\otimes$.

4 Replanning Using the Plan Operators

Although these operators are sufficient to describe all necessary changes, in real planning situations we will need a more refined approach. In this section we will describe how both addition and deletion can be described by iterative plan change processes using a library of plan schemes. This section will show how these operators can be combined in a replanning algorithm. Recall that our problem is to change a plan P in an inadequate triple (R, P, Gs) to a plan P' such that (R, P', Gs) is adequate. It can be proven that the three mentioned operators are sufficient to transform one plan into another, given a suitable plan scheme library. This section will show how these operators can be combined in a replanning algorithm. Recall that, initially, there is a plan P that satisfies certain goals Gs. Because of some events, this goal set Gs is changed to a goal set $Gs' = (Gs \setminus G_D) \cup G_A$, i.e. some of the goals are no longer needed and can be removed (G_D), while others must be newly satisfied (G_A). Proposition 1 states that we can transform the current plan P to the desired plan P' by using one addition and one deletion: $P' = (P \ominus Ps_1\sigma_1) \oplus Ps_2\sigma_2$. In practice however, it is very unlikely that the schemes Ps_1 and Ps_2 occur in our plan scheme library. Instead, we have to search for a sequence of smaller transformations, that together transform the plan completely. This leads to an iterative search procedure in which we, given the current situation (R, P, Ers), try to satisfy a

Algorithm 1. HOWTO

HOWTO(plan P, plan scheme Ps)
1. **if** *all outputs of* Ps *are available in* P **then**
   return$\langle \emptyset, \emptyset \rangle$
2. **let** $Ps' = Ps$
3. **let** $Ers = \emptyset$
4. **for** *all skills* $s \in Ps$ *that produce outputs of* Ps *that are not available in* P **do**
   4.1. **let** $\langle Ps', Ers' \rangle =$ CHECK_SKILL(s, P, Ps')
   4.2. **let** $Ers = Ers \cup Ers'$
5. **return**$\langle P \oplus Ps', Ers \rangle$

CHECK_SKILL (skill s, plan P, plan scheme Ps)
1. **if** s *is marked* **then**
   1.1. **return** $\langle Ps, \langle \emptyset, \emptyset \rangle \rangle$
2. **else**
   2.1. MARK(s)
3. **let** $\langle Rs, C \rangle =$GET_CONSTRAINTS(s)
4. **let** R *be the set of resources* r*, such that* $\exists rs \in Rs \cdot r \models rs \wedge C(r)$ *is satisfiable*
5. **let** $R' \subseteq R$ *be the set of resources, such that* $R' \models Rs' \wedge C(R')$ *is satisfiable, where* $R' \subseteq R$ *is the largest subset of* R *that can be formed this way.*
6. **let** $Rs^* = Rs - Rs'$
7. **let** $Ps' = Ps$*, with all skills removed that are used solely for the production of resources from* $Rs - Rs^*$
8. **if** $Rs^* = \emptyset$ **then**
   8.1. **return** $\langle Ps', \langle \emptyset, \emptyset \rangle \rangle$
9. **else**
   9.1. **let** $Ers = \langle \emptyset, \emptyset \rangle$
   9.2. **for** *all skills* s' *that produce resource schemes of* Rs^* **do**
      9.2.1. **let** $\langle Ps', Ers' \rangle =$ CHECK_SKILL(s', P, Ps')
      9.2.2. **let** $Ers = Ers \cup Ers'$
   9.3. **return**$\langle Ps', Ers \rangle$

subset Er of the needed resources Ers using a plan scheme Ps. As we have seen in Example 1, the addition of a plan scheme may introduce new input resources R_{req}. Therefore, during each iteration, the current situation is changed to the situation $(R, P \oplus Ps\sigma, Ers')$, where $Ers' = (Ers \setminus Er) \cup R_{req}$. After satisfying the new goals by adding plan schemes, the plan may contain actions that are not used for goal production. These have to be identified and removed using the $\ominus$-operator. The remainder of this section will discuss the functions to adapt the current plan using plan schemes in a library.

HOWTO: Adding Plan Schemes. In cases where removal of parts of the existing plan is not necessary, we may try to add plan schemes iteratively in order to satisfy one or more resources that are needed in the current situation. The plan schemes may be instantiated in different ways, affecting the way the plan is adapted. To perform an iteration step we introduce a function HOWTO to find out how to instantiate a given plan scheme Ps in a most efficient way, i.e. to find a substitution σ such that the number of new input resources to the plan is minimized. The HOWTO procedure (which is outlined in Algorithm 1) works as follows: Given the current plan P, satisfying goals Gs using initial resources R, i.e. $R \models_P G\theta$, it takes a goal g that is to be provided. HOWTO then computes an instantiation of Ps and an extended resource scheme Ers such that $R \cup Ers\tau \models_{P \oplus Ps\sigma} (G \cup g)\tau$, for some substitution τ. HOWTO then constructs the new plan $P' = P \oplus Ps\sigma$ and returns a tuple $\langle P', Ers \rangle$ consisting of the new plan and the missing resources Ers. Note that a new iteration step might consist in selecting some missing resource scheme from Ers as a goal and so on.

OVERLAP: Replacing Plan Schemes. Another function to obtain information about how to apply a single plan scheme during one iteration of the search is called OVERLAP and will be used to find a sub plan P' of P that can be removed in order to efficiently add an instantiated plan scheme $Ps\sigma$. This information is then used to implement the $\otimes$-operator, resulting in a plan $P'' = (P \ominus P') \oplus Ps\sigma$.

Like HOWTO, OVERLAP is given the current plan and a plan scheme Ps. It returns either *failure*, if it cannot find a suitable subplan of P to remove to make room for Ps, or a new plan in which Ps is applied to the plan using $\otimes$. The OVERLAP-procedure starts by locating which resources r_o are present in the plan that can also be provided by the plan scheme. These resources are the starting point for determining if part of the plan can be *replaced* by the plan scheme. From these resources, we search backwards in the plan to see if we can find resources r_i that correspond to input resource schemes of Ps. In short, the actions between r_i and r_o can be replaced by Ps.[8] If no such actions can be found, OVERLAP returns *failure*. After determining which subplan P' can be removed and which instantiation σ of the plan scheme to use, OVERLAP removes P' and uses the $\oplus$-operator on the plan $P \ominus P'$ and $Ps\sigma$ to obtain $P'' = P \otimes (P', Ps\sigma)$ i.e., the plan P where P' has been replaced by $mathitPs\sigma$. It also computes an extended resource scheme *Ers* that describes the new input resources that are to be provided. Then OVERLAP constructs a new plan $P'' = P \otimes (P', Ps\sigma)$ and returns the tuple $\langle P'', Ers \rangle$ consisting of the new plan P' and the missing resources *Ers*.

REMOVE_ACTIONS: Removing Obsolete Actions. After we have found a plan that satisfies the current set of goals, the plan still may contain actions that are used to produce actions that are not used to satisfy goals. These obsolete actions are removed by the function REMOVE_ACTIONS which will determine the actions of a plan P that can be removed. This is done by examining all actions a in the plan, and if $out(a) \subseteq free(P)$, then a can be removed using the $\ominus$-operator. Of course, removing a has implications for any actions a' for which $out(a') \cap in(a) \neq \emptyset$, i.e., actions that produce the resources that a consumed. An efficient scheme has to be developed for examining a plan for obsolete actions. Such scheme works as follows: We keep a list *obs* of resources that are not needed anymore. Initially, this list consists of the resources that satisfy the obsolete goals. One by one, the resources $r \in obs$ are removed from *obs* and examined: If the action a that produces r only produces unused resources, then a is removed and the resources $in(a)$ are added to *obs*. This continues until $obs = \emptyset$. The resulting plan has no actions of which products are not used. However, in some instances the plan can be further optimized by using some of the resources that have just become free. In these cases, plan merging techniques such as [3] can be used to optimize it.

Heuristic Search. We have combined the functions described above to implement a best-first search replanning algorithm to transform a plan into a plan

[8] Though this is quite a simplification of things, we omit the details for brevity.

that satisfies a new goal set G'. First, the current plan of the agent is expanded by using HOWTO and OVERLAP on all possible plan schemes. This produces a set of partial plans, from which we select a cheapest one. This plan is removed from the set and expanded. This produces a new set of partial plans, consisting of the partial plans of the first expansion, combined with those of the second expansion. Again, we select a cheapest partial plan and expand it. This continues until we have found a solution. When we have found a solution, we have found a solution, i.e. a plan satisfying G'. Finally, this plan is simplified by using REMOVE_ACTIONS.

5 Experimental Results

This section presents some preliminary results, as reported in [13]. The experiments were conducted as follows: First, we generated a plan with Blackbox for the four problems. Two of the problems (number 3 and 4) were from the AIPS planning competition, the two others were constructed by ourselves in ARPF. These consisted of a graph of 9 locations in which 2 trucks drive around, that can each move two loads simultaneously. A STRIPS translation of these plans was made to feed the Blackbox planner.

After making the initial plans, we randomly selected new goals for the AIPS problems and constructed a number of hard goals for our own problems. Then we ran Blackbox again for the new problems, and our own algorithm with the initial plan. Table 1 contains an overview of some running times for the four different problems. For different queuing strategies (different cost functions) we ran a series of 10 tests to see what the average time to complete was, and which size the resulting plans had. The *minimum* and *maximum* times in the table refer to the average time of the fastest and slowest strategy, the *average* time is the average of the average time for each strategy. As one can see, even in the worst cases, we performed better than Blackbox (whose time was also averaged over 10 runs). Note that the STRIPS translation of our framework contains a lot of overhead, which is why the difference is exceptionally large on these problems.

Table 1 also contains the results for the size of the solutions found. Again, the *minimum* and *maximum* sizes refer to the average size of the shortest and longest plans and the *average* size is the average of the average sizes. Here, we perform worse than Blackbox. In some cases, our solution is slightly better, but in general we do not find an optimal solution, considering the number of steps. One advantage we do have, however, is that we remain close to the original plan, where Blackbox sometimes finds a completely different plan.

6 Evaluation

We have remarked that a plan that is created under the assumptions of deterministic effects and sole cause of change may not remain valid in a dynamic environment. Other agents or failing actions may bring an agent into a situation where its plan becomes obsolete or inefficient. If this happens, the agent has to

Table 1. Time (in seconds) required to solve a problem and the size (in actions) of the resulting plan.

time	Prob 1	Prob 2	Prob 3	Prob 4
Minimum	0.35	1.58	0.10	0.20
Average	0.85	5.98	0.46	0.65
Maximum	1.73	9.47	0.73	0.98
Blackbox	233.36	508.30	2.53	43.93

size	Prob 1	Prob 2	Prob 3	Prob 4
Minimum	16.0	20.0	17.6	26.0
Average	18.7	22.5	18.2	26.5
Maximum	31.2	33.3	18.7	27.8
Blackbox	16.0	22.0	18.0	23.0

create a new plan. We proposed the use of generic plan-adjustment operators to adapt the current plan to the new situation. We showed how these operators are defined in the Action and Resource Planning Formalism and showed an iterative algorithm based on them and some initial experimental results with this algorithm

Future work includes a more efficient implementation and the incorporation of an interesting idea called *Sliding Scale of Commitment* (SSC) of Kott and Saks [10]. The principle underlying SSC is based on the observation that in general it is less worse to break a commitment far in the future than one whose deadline is soon. This heuristic adds extra costs to the plan changes the affect commitments in the near future. Furthermore, we are working on an *ordered plan scheme library*, to try more preferred plan schemes first.

Other approaches to plan modification have been proposed. Gerevini and Serina [5] have proposed a system that is based on Graphplan [2]. Their method requires that additional data structures that were used during the planning phase are still available during the replanning process. This also holds for the solution of Hanks and Weld [6], which records the *reasons* for each step that is taken during planning, and for the plan modification theory of Kambhampati [8], which relies on a *validation structure* that is computed during planning. An advantage of our method is that it does not require such additional data structures to remain available.

Ambite and Knoblock [1] also introduce the concept of plan operations. Their *plan rewriting rules* are domain-dependent however. Also, their system only considers complete plans, which greatly reduces the search space. This may mean that a large number of solutions cannot be found, and that the approach will not work once a plan is already broken. This makes it less suitable for use as a replanning method.

Another approach towards robust plan execution is to generate plans that already have measures to handle incidents. These plans have so-called conditional effects (see, e.g. [12]) depending on the actual situation of the world. The problem with this approach is that at forehand all possible contingencies must be enumerated together with the corresponding countermeasures (but see [11] for a discussion of selecting important contingencies).

References

1. J.L. Ambite and C.A. Knoblock. Planning by rewriting: Effiently generating high-quality plans. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 706–713, Providence, Rhode Island, 1997. AAAI Press/MIT Press.
2. A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
3. M.M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging. *Submitted to the Annals of Mathematics and Artificial Intelligence*, 2001.
4. P. Doherty and J. Kvarnstrom. Talplanner: An empirical inverstigation of a temporal logicbased forward chaining planner, 1999.
5. A. Gerevini and I. Serina. 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, 2000.
6. S. Hanks and D.S. Weld. A domain-independent algorithm for plan adaptation. *Journal of AI Research*, 2:319–360, 1995.
7. J. Hoffmann and B. Nebel. Fast plan generation trough heuristic search, 2000.
8. S. Kambhampati. A theory of plan modification. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 176–182, Boston, Massachusetts, USA, 1990. AAAI Press/MIT Press.
9. H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the workshop on planning as combinatorial search, held in conjunction with AIPS'98*, Pittsburgh, PA, 1998.
10. A. Kott and V. Saks. Continuity-guided regeneration: An approach to reactive replanning and rescheduling. In *Proceedings of the Florida AI Research Symposium*, 1996.
11. N. Onder. *Contingency Selection in Plan Generation*. PhD thesis, University of Pittsburgh, 1999.
12. L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.
13. R.P.J. van der Krogt. *Replanning methods in a skill-based framework*. Master's thesis, Delft University of Technology, Delft, August 2000.
14. D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.