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. However, since most agents live in a dynamic environment, they will often be confronted with situations in which their plans are no longer feasible or optimal. 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 so-called *resources* and *actions* to represent the planned activities of an agent. A (complete) algorithm is introduced which can be used to replan the activities, taking the new environment into account.

1 Introduction

Agents often have to achieve a number of given goals without a predefined way to accomplish these goals. Therefore, they have to make a plan that consists of a number of actions that, starting with the current situation, brings the agents to the 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 [3], FF [6]. Most of such planning systems assume that the agent is the *sole cause of change* and that the actions have *deterministic effects* (see [9] for a discussion of planning systems and their assumptions). While these assumptions are necessary to make the planning process more or less tractable, they usually no longer hold when an agent starts executing its plan in the real world. For instance, the agent will not likely be the only source of interactions with the world, violating the sole cause of change assumption. In these situations, agents have to cope with unpredictable situations, 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. Besides being inefficient, it might also violate agreements that have been made with other agents. Therefore, we propose a specialized plan revision method (like in [4][5]) to adapt the current plan to a plan which takes into account the new situation. These revision methods make use of additional information they gathered during the planning process, such as the reasons for adding certain steps. Our approach does not need such additional information. It requires the presence of a library of *plan fragments*, i.e. small plans that define the services an agent can provide.This library may contain a

* An extended version of this paper has been submitted to the European Conference on Planning.

set of possible (pre-compiled) plan repairs, but can also come from other agents that "advertise" their services.

The remainder of this 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 three 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 [2]. 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, $truck(5 : id, \mathrm{StL} : loc, 15.^{00} : time)$ is the resource that describes truck number 5 being in St. Louis at $15.^{00}$ hrs. The set of all resources is denoted with $\mathcal{R}$.

A *resource scheme* rs is the set of all resources with the same predicate symbol, sharing a number of attributes. Such a resource scheme is specified by giving the type of the resources and the set of values for their attributes, where each value is either (i) a variable, meaning that this attribute may differ between two resources in the set, or (ii) a ground value, requiring that this attribute has the same value for all resources in the set. To distinguish between ground values and variables, we require that a '?' is placed in front of each variable name, e.g. $?x$ denotes a variable x. For example, the resource scheme $truck(?i : id, \mathrm{StL} : loc, ?t : time)$ 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 $rs\theta = r$. Such a substitution changes each occurrence of a variable with its value. 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 $Rs\theta \subseteq R$. A substitution θ is said to be resource-identity preserving if $\forall rs_1, rs_2 \in Rs \cdot rs_1 \neq rs_2 \rightarrow rs_1\theta \neq rs_2\theta$. 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,

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

denotes a truck and load in St. Louis at the same time. A resource set R satisfies an extended resource scheme $\langle Rs, C\rangle$, if $Rs\theta \subseteq R$, for some resource-identity preserving substitution θ and $\models C\theta$, i.e. all constraints are true for θ.

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 that are produced by a, Rs_1 is the set of resources schemes that are consumed by a and C are the constraints on Rs_1. For example,

$$driveStL : \{truck(?i : id, \text{StL} : loc, ?t + d(?l, \text{StL}) : time)\} \leftarrow$$
$$\langle \{truck(?i : id, ?l : loc, ?t : time)\}, \{?t > 7.^{00}, ?l \neq \text{StL}\}\rangle$$

is an action to drive a truck to St. Louis. From a truck at some location $l \neq$ StL at time $t > 7.^{00}$, it is possible to "produce" a truck in St. Louis at time $t+d(?l, \text{StL})$. The set of input resources is denoted with $in(s) = Rs_1$, the output resources are denoted with $out(s) = Rs_2$. 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 an instantiation θ changing each occurrence p of a predicate symbol in $in(s)$ and $out(s)$ 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.

Goals and Plans The *goals* of an agent 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 the constraints on G.

A *plan* is a sequence of actions that, starting with some initial resources, produces a number of resources that satisfy the goal conditions. A plan is represented by a bi-partite Directed Acyclic Graph $P = \langle N_R \cup N_A, E\rangle$, where N_R is a set of resource nodes, N_A a set of action nodes n_a where $a \in A$ and E a set of arcs. For any two nodes $r \in N_R$ and $n_a \in N_A$, (r, n_a) means that resource r is used by an application of action a, and (n_a, r) means that resource r is produced by an application of a. The set of *input* resources of P is denoted by $in(P)$, whereas $out(P)$ will refer to the set of final products of 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 resources R and a substitution θ that assigns each goal scheme with a resource in $out(P)$, denoted by $R \models_P Gs\theta$, if $in(P) \models_P Gs\theta$ and $R \supseteq in(P)$. The triple (R, P, Gs) is called *valid*, if $R \models_P Gs\theta$ for some θ. It is called *partial*, if there exists an R' such that $R' \models_P Gs\theta$, but $R \not\models_P Gs\theta$. A plan P is called *valid for* Gs, if $R_A \models_P Gs\theta$ for some initial resources R_A. It is called *partial for* Gs, if for each R' such that $R' \models_P Gs\theta$ the resources of R' are not all contained in the initial resources R_A.

A *plan fragment* is a special kind of plan to define *services* an agent can provide. Unlike plans, plan fragments may contain resource schemes and must be instantiated like actions. Plan fragments can be found by analyzing the previous plans of the agents, or given in the domain description. Analogously to actions, for a plan fragment pf, the substitution σ assigns each variable in the plan fragment a value, thus substituting grounded resources for the resource schemes. A fully instantiated fragment $pf\sigma$ is a plan.

3 Plan operators

An agent constructs a plan to satisfy some predetermined goals Gs. However, as we have seen, a plan P may fail during execution because of an unanticipated event, such as a resource that was not available as expected or because one of the actions failed to produce the required resources. The problems the agent might encounter, can all be described in terms of a changed set of goals. For example, if an action fails to produce some resources, these resources can be thought of as new goals that are to be satisfied to ensure that the plan can be executed properly. Once a goal set has changed, the plan has to be adapted to satisfy the new set of goals. We thus concentrate on the following problem:

> Given an original plan P, satisfying a set G of goals while using a set of resources R, we must adapt the plan P to a plan P' such that it satisfies a new goal set G', using the same resources R, i.e., $R \models_{P'} G'$.

Intuitively, we can change a plan in two ways: (i) the plan can be *extended* by adding one or more actions by the so-called *addition operator* $\oplus$, and (ii) the plan can be *reduced* by removing one or more actions by the *deletion operator* $\ominus$. These two concepts can be combined to *replace* some actions in the plan by others using the *replacement operator* $\otimes$. Therefore, in order to adapt a plan P with goals G using initial resources R_A, denoted by the triple (R_A, P, G), we introduce these three basic operations of addition, deletion and replacement.

Addition Two plans can be combined to form a bigger plan, much like the union of two sets form a bigger set. The addition of two plans is denoted by $P' = P_1 \oplus P_2$.

Definition 1. *Let $P_1 = (N_1, E_1)$ and $P_2 = (N_2, E_2)$ be plans. Then, P_1 and P_2 are **compatible**, if*

- *for every node $r \in N_{R_1} \cap N_{R_2}$, if there are edges $(r, a) \in E_1 \wedge (r, b) \in E_2$ or $(a, r) \in E_1 \wedge (b, r) \in E_2$, then $a = b$; That is, if P_1 and P_2 have common resources, then these resources are neither produced nor consumed by different actions.*
- *let $out_i(a)$ denote $out(a)$ in plan i and $in_i(a)$ denote $in(a)$ in plan i $(i = 1, 2)$, then for every action $a \in N_{A_1} \cap N_{A_2}$, $out_1(a) = out_2(a)$ and $in_1(a) = in_2(a)$; That is, if P_1 and P_2 have actions in common, then these actions consume and produce the same resources.*

For two plans P_1 and P_2 that are compatible, $P_1 \oplus P_2$ is defined as $P' = (N_1 \cup N_2, E_1 \cup E_2)$. To add a plan fragment pf to a plan P, we first have to instantiate it using a substitution σ. This σ is chosen so that:

- P and $pf\sigma$ are compatible,
- $P' = P \oplus pf\sigma$ is acyclic, and
- For every substitution τ such that P and $pf\tau$ are compatible, $|in(P \oplus pf\tau)| \geq |in(P \oplus pf\sigma)|$; no other substitution exists that has less new input resources.

Deletion Deleting a plan P_2 from a plan P_1 is denoted as $P' = P_1 \ominus P_2$, and is done by removing the common part of P_1 and P_2 from the former plan. For two compatible plans $P_1 = (N_{A_1} \cup N_{R_1}, E_1)$ and $P_2 = (N_{A_2} \cup N_{R_2}, E_2)$, $P' = P_1 \ominus P_2$ is defined as $P' = (N_A \cup N_R, E)$, where:

- $N_A = (N_{A_1} \cup N_{A_2}) \setminus (N_{A_1} \cap N_{A_2})$; That is, the common actions of both plans are removed.
- Let R be the set of resources r in $N_{R_1} \cup N_{R_2}$ such that for all edges (a, r) and (r, a) in $E_1 \cup E_2$, $a \in (N_{A_1} \cap N_{A_2})$. Then $N_R = (N_{R_1} \cup N_{R_2}) \setminus R$; That is, the resources that belong to the removed actions are removed.
- Let D be the set of arcs (a, r) or (r, a), where $a \in N_{A_1} \cap N_{A_2}$ or $r \in R$. Then $E = (E_1 \cup E_2) \setminus D$; That is, the edges that connect removed actions or resources are deleted.

Replacement The replacement of a plan fragment pf' by a plan fragment pf in a plan P is denoted by $P' = P \otimes (pf', pf)$. This is done by first removing the (instantiated) plan fragment $pf'\sigma$ and then adding the instantiated $pf\tau$. Formally, $P' = (P \ominus pf'\sigma) \oplus pf\tau$. Note that:

- The substitutions σ and τ should be chosen such that P and $pf'\sigma$ are compatible, as well as $(P \ominus pf'\sigma)$ and $pf\tau$.
- This definition does not require that pf is placed at the location of which pf' was removed; in fact, pf' and pf need not have anything in common.

4 Replanning using the plan operators

It can be proven that the three mentioned operators are sufficient to transform one plan into another, given a suitable plan fragment 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). We will now show how we can use the operators of the previous section to incorporate or remove goals.

4.1 New goals

In case the agent has to satisfy a new goal, there are two things the agent can do if its plan does not already provide the required goals: Either it can *add* new actions to the plan that satisfy the new goal, or the agent can *replace* one or more actions with more powerful ones that satisfy the goal, while still satisfying the original goals. Clearly, these concepts correspond with the $\oplus$- and $\otimes$-operators.

Satisfying new goals by adding actions An agent may try to satisfy additional goals by adding new actions to its plan using the $\oplus$-operator. In order to get to know how to apply a single plan fragment, we introduce the HOWTO function. HOWTO will be used to find out which part a plan fragment pf has in common with the plan P, and which new input resources will be needed. This information is then used to add pf to the plan, using the $\oplus$-operator.

HOWTO is given the current plan P, a selected goal g that is to be satisfied and a plan fragment pf (which produces g) to apply. By checking the actions and resource schemes in pf, it is able to compute the instantiated plan fragment $pf\sigma$, which minimizes the number of input resources for the plan $P' = P \oplus pf\sigma$, while establishing a resource $r \in out(P')$ such that $r \models g\theta$. At the same time, the extended resource scheme Ers that describes the resources $in(P') \setminus in(P)$ is determined, i.e., the new input resources to the plan. In other words, given a

plan P, satisfying goals G and a goal g that is to be provided, HOWTO computes an instantiated plan fragment $pf\sigma$ and an extended resource scheme Ers such that $R_A \cup Ers\tau \models_{P \oplus pf\sigma} (G \cup g)\theta$, for some substitutions θ and τ and initial resources R_A. HOWTO then constructs the new plan $P' = P \oplus pf\sigma$ and returns a tuple $\langle P', Ers \rangle$ consisting of the new plan and the missing resources Ers.

Satisfying new goals by replacing actions In order to satisfy new goals, the agent may also consider replacing certain actions by others. Again, we will introduce a function to obtain information about how to apply a single plan fragment. This function will be called OVERLAP and will be used to find a sub plan P' of P that can be removed to efficiently add an instantiated plan fragment $pf\sigma$. This information is then used to implement the $\otimes$-operator, resulting in a plan $P'' = (P \ominus P') \oplus pf\sigma$.

Like HOWTO, OVERLAP is given the current plan and a plan fragment pf. It returns either *failure*, if it cannot find a suitable overlap between the plan P and pf, or a new plan in which pf is applied to the plan using $\otimes$. The OVERLAP-procedure starts by locating which resources o are present in the plan that the plan fragment can provide. These resources are the starting point for determining if part of the plan can be replaced by the plan fragment. From these resources, we search backwards in the plan to see if we can find resources i that correspond to input-resource schemes of pf. In short, the actions between i and o can be replaced by pf.[1] If no such actions can be found, OVERLAP returns *failure*. After determining which sub plan P' can be removed and which instantiation σ of the plan fragment to use, OVERLAP removes P' and uses the $\oplus$-operator on $P \ominus P'$ and $pf\sigma$ to obtain $P'' = P \otimes (P', pf\sigma)$. 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 $P'' = P \otimes (P', pf\sigma)$ and returns the tuple $\langle P'', Ers \rangle$ consisting of the new plan P' and the missing resources Ers.

4.2 Obsolete goals

If there are one or more goals that do not have to be satisfied anymore, the plan may contain actions that are now obsolete, because they produce only resources that are used for satisfying the obsolete goals. These actions may be removed, as they no longer have a purpose to serve.

We will use a function REMOVE_ACTIONS which will determine which of the actions of a plan P can be removed because they are used solely to produce obsolete goals. This is done by examining all actions a in the plan, and if $out(a) \subseteq free(P)$, then a produces only unused resources and can be removed using the $\ominus$-operator. Of course, removing a has implications for any actions a' for which $out(a') \cap in(a) \neq \emptyset$, actions that produce the resources that a consumed. An efficient scheme has been developed for examining a plan for obsolete actions. This scheme works as follows: We keep a list obs of resources that are not needed anymore. Initially, this list is filled with 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 $obs = \emptyset$.

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

4.3 Heuristic search

We have combined the functions described above to implement a best-first search replanning algorithm to transform a plan into a plan that satisfies a new goal set G'. First, we expand the current plan of the agent by using HOWTO and OVERLAP on all possible plan fragments. This produces a set of partial plans, from which we pick the 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 pick the cheapest partial plan and expand it. This continues until we have found a solution. When we have found a solution, we have a plan that still satisfies the obsolete goals, the ones that the agent no longer needs. This plan is simplified by using REMOVE_ACTIONS. This produces a plan that satisfies G'.

5 Experimental Results

This section presents some preliminary results, as reported in [8]. 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.

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

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 create a new plan. We proposed the use of generic plan-adjustment operators to adapt the current plan for the new situation. We showed how these operators are defined in the Action and Resource Planning Formalism and showed implementations of these operators. We also showed a combination of these operations, that provides a complete algorithm for replanning 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 [7]. 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 fragment library*, to try more preferred plan fragments first.

Other approaches to plan modification have been proposed. Gerevini and Serina [4] have proposed a system that is based on Graphplan [1]. 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 [5], which records the *reasons* for each step that is taken during planning. An advantage of our method is that it does not require such additional data structures to remain available.

References

1. A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
2. M.M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging, 2001.
3. P. Doherty and J. Kvarnstrom. Talplanner: An empirical inverstigation of a temporal logicbased forward chaining planner, 1999.
4. A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proc. of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pages 112–121, 2000.
5. S. Hanks and D.S. Weld. A domain-independent algorithm for plan adaptation. *Journal of AI Research*, 2:319–360, 1995.
6. J. Hoffmann and B. Nebel. Fast plan generation trough heuristic search, 2000.
7. A. Kott and V. Saks. Continuity-guided regeneration: An approach to reactive replanning and rescheduling. In *Proc. of the Florida AI Research Symposium*, 1996.
8. R.P.J. van der Krogt. Replanning methods in a skill-based framework. Master's thesis, Delft University of Technology, Delft, August 2000.
9. D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.