Contingent Planning with Goal Preferences*

Dmitry Shaparau
ITC-IRST, Trento, Italy
shaparau@irst.itc.it

Marco Pistore
University of Trento, Italy
pistore@dit.unitn.it

Paolo Traverso
ITC-IRST, Trento, Italy
traverso@irst.itc.it

Abstract

The importance of the problems of contingent planning with actions that have non-deterministic effects and of planning with goal preferences has been widely recognized, and several works address these two problems separately. However, combining conditional planning with goal preferences adds some new difficulties to the problem. Indeed, even the notion of optimal plan is far from trivial, since plans in non-deterministic domains can result in several different behaviors satisfying conditions with different preferences. Planning for optimal conditional plans must therefore take into account the different behaviors, and conditionally search for the highest preference that can be achieved. In this paper, we address this problem. We formalize the notion of optimal conditional plan, and we describe a correct and complete planning algorithm that is guaranteed to find optimal solutions. We implement the algorithm using BDD-based techniques, and show the practical potentialities of our approach through a preliminary experimental evaluation.

Introduction

Several works deal with the problem of *contingent planning*, i.e., the problem of generating conditional plans that achieve a goal in non-deterministic domains, see, e.g., (Hoffmann & Brafman 2005; Rintanen 1999; Cimatti *et al.* 2003). In these domains, actions may have more than one outcome, and it is impossible for the planner to know at planning time which of the different possible outcomes will actually take place at execution time. The importance of the problem of *planning with goal preferences* has also been widely recognized, see, e.g., (Brafman & Chernyavsky 2005; Briel *et al.* 2004; Smith 2004; Brafman & Junker 2005). In planning with preferences, the user can express preferences over goals and situations, and the planner must generate plans that meet these preferences. More and more research is addressing these two important problems separately, but the problem of *contingent planning with goal preferences* has not been addressed yet. This is clearly an interesting problem, and most applications need to deal both with non-determinism and with preferences. However, providing planning techniques that can address the combination of the two aspects requires to solve some difficulties, both conceptually and practically.

Conceptually, even the notion of optimal plan is far from trivial. Plans in non-deterministic domains can result in several different behaviors, some of them satisfying conditions with different preferences. Planning for optimal conditional plans must therefore take into account the different behaviors, and conditionally search for the highest preference that can be achieved. In deterministic domains, the planner can plan for (one of) the most preferred goal, and in the case there is no solution, it can iteratively look for less preferred plans. This approach is not possible with non-deterministic domains. Since actions are non-deterministic, the planner will know only at run-time whether a preference is met or not, and consequently a plan must interleave different levels of preferences, depending on the different action outcomes. In this setting, devising planning algorithms that can work *in practice*, with large domains and with complex preference specifications, is even more an open challenge.

In this paper we address this problem, both from a conceptual and a practical point of view. Differently from the decision theoretic approach taken, e.g., in (Boutilier, Dean, & Hanks 1999) we allow for an explicit and qualitative model of goal preferences. The model is *explicit*, since it is possible to specify explicitly that one goal is better than another one. It is *qualitative*, since preferences are interpreted as an order over the goals. Moreover, the model takes into account the non-determinism of the domain. If we specify that a goal g_1 is better than g_2, we mean that the planner should achieve g_1 in all the cases in which g_1 can be achieved, and it should achieve g_2 in all the other cases.

We define formally the notion of optimal conditional plan, and we devise a planning algorithm for the corresponding planning problem. The algorithm is complete and correct, i.e., it is guaranteed to find only optimal solutions, and to find an optimal plan if it exists. We have designed the algorithm to deal in practice with complex problems, i.e., with large domains and complex goal preference specifications. Intuitively, the underlying idea is to generate first "universal plans" that are guaranteed to achieve least preferred goals.

*This work is partially funded by the MIUR-FIRB project RBNE0195K5, "Knowledge Level Automated Software Engineering", by the MIUR-PRIN 2004 project "Advanced Artificial Intelligence Systems for Web Services", and by the EU-IST project FP6-016004 "Software Engineering for Service-Oriented Overlay Computers".

The plan is "universal" in the sense that it is not restricted to a set of initial states, and the algorithm finds instead the set of states from which a plan exists. We can thus store once for all and then re-use the "recovery" solution for all those sets of states in which a plan for a goal with higher preference cannot be found. We then plan for the next more preferred goal, by relaxing the problem and allowing for solutions that either satisfy it or result in states where plans for the least preferred goal can be applied. We iterate the procedure by taking into account the order of preferences.

The algorithm works on sets of states, and we implement it using BDD-based symbolic techniques. We perform a preliminary set of experimental evaluations with some examples taken from two different domains: robot navigation and web service composition. We evaluate the performances w.r.t. the dimension of the domain, as well as w.r.t. the complexity of the goal preference specification (increasing the number of preferences). The experimental evaluation shows the potentialities of our approach.

The paper is structured as follows. We first review basic definitions of planning in non-deterministic domains. Then we propose a conceptual definition of conditional planning with goal preferences, we describe the planning algorithm for the new goal language, and evaluate it. Finally, we discuss related works and propose some concluding remarks.

Background

The aim of this section is to review basic definitions of planning in non-deterministic domains which we use in the rest of the paper. All of them are taken, with minor modifications, from (Cimatti *et al.* 2003).

We model a *(non-deterministic) planning domain* in terms of *propositions*, which characterize system *states*, of *actions*, and of a *transition relation* describing system evolution from one state to possible many different states.

Definition 1 (Planning Domain) *A planning domain $\mathcal{D}$ is a 4-tuple $\langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ where*

- *$\mathcal{P}$ is the finite set of basic propositions,*
- *$\mathcal{S} \subseteq 2^{\mathcal{P}}$ is the set of states,*
- *$\mathcal{A}$ is the finite set of actions,*
- *$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation.*

We denote with $\text{Act}(s) = \{a : \exists s'.\mathcal{R}(s, a, s')\}$ the set of actions that can be performed in state s, and with $\text{Exec}(s, a) = \{s' : \mathcal{R}(s, a, s')\}$ the set of states that can be reached from s performing action $a \in \text{Act}(s)$.

An example of the planning domain, that will be used throughout the paper, is defined as follows.

Example 1 *Consider the simple robot navigation domain represented in Figure 1. It consists of a building of 6 rooms and of a robot that can move between these rooms performing the actions described in Figure 1. Notice that action "goRight" performed in "Hall" moves the robot either to "Room 1" or to "Room 2" non-deterministically. Notice also that the door between "Room 3" and "Store" might be locked, therefore the action "goRight" performed in "Room 3" is also non-deterministic.*

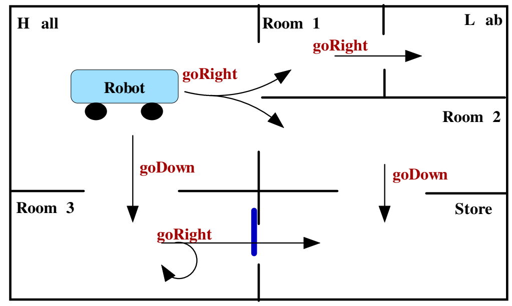


Figure 1: A simple domain

The state of the domain is defined in terms of fluent `room`*, that describes in which room the robot is currently in, and of fluent* `door`*, that describes the state of the door between "Room 3" and "Store", which is initially "Unknown", and becomes either "Open" or "Locked" when the robot tries to move from "Room 3" to "Store" (only in the former case the robot will successfully reach the "Store").*

We represent plans by state-action tables, or policies, which associate to each state the action that has to be performed in such state. As discussed in (Cimatti *et al.* 2003), plans as state-actions tables enable to encode conditional behaviors.

Definition 2 (Plan) *A plan π for a planning domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ is a state-action table which consists of a set of pairs $\{\langle s, a \rangle : s \in \mathcal{S}, a \in Act(s)\}$*

We denote with $\text{StatesOf}(\pi) = \{s : \langle s, a \rangle \in \pi\}$ the set of states in which plan π can be executed.

We describe the possible executions of a plan with an execution structure.

Definition 3 (Execution Structure) *Let π be a plan for a planning domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$. The execution structure induced by π from the set of initial states $\mathcal{I} \subseteq \mathcal{S}$ is a tuple $K = \langle Q, T \rangle$, where $Q \subseteq \mathcal{S}$ and $T \subseteq \mathcal{S} \times \mathcal{S}$ are inductively defined as follows:*

1. *if $s \in \mathcal{I}$, then $s \in Q$, and*
2. *if $s \in Q$ and $\exists \langle s, a \rangle \in \pi$ and $s' \in Exec(s, a)$, then $s' \in Q$ and $T(s, s')$.*

A state $s \in Q$ is a terminal state of K if there is no $s' \in Q$ such that $T(s, s')$.

A planning problem is defined by a planning domain $\mathcal{D}$, a set of initial states $\mathcal{I}$ and a set of goal states $\mathcal{G}$.

Definition 4 (Planning Problem (without preferences)) *Let $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ be a planning domain. A planning problem for $\mathcal{D}$ is a triple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ and $\mathcal{G} \subseteq \mathcal{S}$.*

Example 2 *A planing problem for the domain of Example 1 is the following:*

- $\mathcal{I} : \; room = Hall \wedge door = Unknown$
- $\mathcal{G} : \; room = Store$

Notice that we represent sets of states as boolean formulas on basic propositions. The intuition of the goal is that the robot should move from "Hall" to "Store".

Intuitively, a solution to a planning problem is a plan which can be executed from any state in the set of initial states $\mathcal{I}$ to reach states in the set of goal states $\mathcal{G}$. Due to the non-determinism in the domain, we need to specify the "quality" of the solution by applying additional restrictions on "how" the set of goal states should be reached. In particular we distinguish *weak* and *strong* solutions. A weak solution does not guarantee that the goal will be achieved, it just says that there exists at least one execution path which results in a terminal state that is a goal state. A strong solution guarantees that the goal will be achieved in spite of non-determinism, i.e., all execution paths of the strong solution always terminate and all terminal states are in a set of goal states.

Definition 5 (Strong and Weak Solutions)
Let $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ and $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ be a planning domain and problem respectively. Let π be a plan for $\mathcal{D}$ and $K = \langle Q, T \rangle$ be the corresponding execution structure.

- *π is a strong solution to P if all the paths in K are finite and their terminal states are in $\mathcal{G}$.*
- *π is a weak solution to P if some of the paths in K terminate with states in $\mathcal{G}$.*

We call a state-action pair $\langle s, a \rangle \in \pi$ *strong* if all execution paths from $\langle s, a \rangle$ terminate in the set of goal states. We call it *weak* if it is not strong, and at least one execution path from $\langle s, a \rangle$ terminates in the set of goal states. Intuitively, a weak solution contains at least one weak state-action pair, while a strong solution consists of strong state-action pairs only.

Example 3 *Consider the following plan π_1 for the domain of Example 1:*

state	*action*
$room = Hall$	*goDown*
$room = Room3 \wedge door = Unknown$	*goRight*

Plan π_1 causes the robot to move down to "Room 3" and after that move right to "Store". It is a weak solution for the planning problem from Example 2, indeed the `door` *can non-deterministically become "Locked" in action "goRight", in which case the plan execution terminates without reaching the "Store". This planning problem has indeed no strong solutions at all.*

In (Cimatti *et al.* 2003), an efficient BDD-based algorithm is presented to solve strong (and weak) planning problems. The key feature of the algorithm is that it builds the solution *backwards*, starting from the goal states, and adding a pair $\langle s, a \rangle$ to the plan π only if the states in $\text{Exec}(s, a)$ are either goal states or already included in $\text{StatesOf}(\pi)$. This approach has the advantage that a state is added to the plan only if we are sure that the goal can be achieved from that state, and hence no backtracking is necessary during the search.

Planning for goals with preferences

This section addresses the problem of finding solutions to the problem of planning with preferences. We start by giving a definition of a reachability goal with preferences and of plans satisfying such a goal. Then we will move to the core of the paper: plan optimality.

Definition 6 (Reachability Goal with Preferences)
A reachability goal with preferences $\mathcal{G}_{list}$ is an ordered list $\langle g_1, ..., g_n \rangle$, where $g_i \subseteq S$. The goals in the list are ordered by preferences, where g_1 is the most preferable goal and g_n is the worst one.

For simplicity, we consider a totally ordered list of preferences, even if the approach proposed in the paper could be easily extended to the case of goals with preferences that are partial orders.

Definition 7 (Planning Problem with Preferences)
Let $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ be a planning domain. A planning problem with preferences for $\mathcal{D}$ is a triple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{list} \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ and $\mathcal{G}_{list} = \langle g_1, ..., g_n \rangle$ is a reachability goal with preferences.

Example 4 *Let us consider a goal with preferences for the domain of Example 1 which consists of two preference goals $\mathcal{G}_{list} = \langle g_1, g_2 \rangle$, where:*

- $g_1 = \{room = Store\}$
- $g_2 = \{room = Lab\}$

The intuition of this goal is that the robot has to move to "Store" or "Lab", but "Store" is a more preferable goal and the robot has to reach "Lab" only if "Store" becomes unsatisfiable. The planning problem from Example 2 can be extended to the planning problem with preferences, defined by the triple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{list} \rangle$.

Plans as state-action tables are expressive enough to be solutions for planning problems for reachability goals with preferences.

Definition 8 (Solution)
A plan π is a solution for the planning problem $P = \langle \mathcal{D}, \mathcal{I}, \{g_1, ..., g_n\} \rangle$ if it is a strong solution to the planning problem $\langle \mathcal{D}, \mathcal{I}, \bigvee_{1 \leq i \leq n} g_i \rangle$.

This definition of plan does not take into account goal preferences. All plans are equally preferable regardless from which goals from the list are satisfied. To overcome this limit, we develop an ordering relation between plans and define a notion of plan optimality.

In the definition of the ordering relation among plans, we have to take into account that, due to non-determinism, different goal preferences can be reached by considering different executions of a plan from a given state. In our approach, we will take into account the "extremal" goal preferences that are reached by executing a plan in a state, namely the goal with the best preference and that with the worst preference. Formally, we denote with $\mathit{pref}(\pi, s)^{\mathit{best}}$ the goal with best preference (i.e., of minimum index) achievable from s: $\mathit{pref}(\pi, s)^{\mathit{best}} = \min\{i : \exists s' \subseteq g_i$ and s', is a terminal state of the execution structure for π that can be reached from $s\}$. The definition of goal $\mathit{pref}(\pi, s)^{\mathit{worst}}$ with worst preference reachable from s is similar. If $s \notin \text{StatesOf}(\pi)$ then $\mathit{pref}(\pi, s)^{\mathit{best}} = \mathit{pref}(\pi, s)^{\mathit{worst}} = -\infty$.

In the following definition, we compare the quality of two plans π_1 and π_2 in a specific state s of the domain. We use an *optimistic behavior assumption*, i.e., we compare the plans according to the goals of best preferences reached by the plans (i.e., $pref(\pi_1, s)^{best}$ and $pref(\pi_2, s)^{best}$). In case the maximum possible goals are equal, we apply a *pessimistic behavior assumption*, i.e., we compare the plans according to the goals of worst precedence (i.e., $pref(\pi_1, s)^{worst}$ and $pref(\pi_2, s)^{worst}$).

Definition 9 (Plans Total Ordering Relation in a State)
Let π_1 and π_2 be plans for a problem P. Plan π_1 is better than π_2 in state s, written $\pi_1 \prec_s \pi_2$, if:

- $pref(\pi_1, s)^{best} < pref(\pi_2, s)^{best}$, *or*
- $pref(\pi_1, s)^{best} = pref(\pi_2, s)^{best}$ *and* $pref(\pi_1, s)^{worst} < pref(\pi_2, s)^{worst}$

If $pref(\pi_1, s)^{best} = pref(\pi_2, s)^{best}$ and $pref(\pi_1, s)^{worst} = pref(\pi_2, s)^{worst}$ then π_1 and π_2 are equivalent in state s, written $\pi_1 \simeq_s \pi_2$.
We write $\pi_1 \preceq_s \pi_2$ if $\pi_1 \prec_s \pi_2$ or $\pi_1 \simeq_s \pi_2$.

We can now define relations between plans π_1 and π_2 by taking into account their behaviors in the common states $S_{common}(\pi_1, \pi_2) = \text{StatesOf}(\pi_1) \cap \text{StatesOf}(\pi_2)$.

Definition 10 (Plans Ordering Relation)
Let π_1 and π_2 be plans for a problem P. Plan π_1 is better than plan π_2, written $\pi_1 \prec \pi_2$, if:

- $\pi_1 \preceq_s \pi_2$ *for all states* $s \in S_{common}(\pi_1, \pi_2)$, *and*
- $\pi_1 \prec_{s'} \pi_2$ *for some state* $s' \in S_{common}(\pi_1, \pi_2)$.

If $\pi_1 \simeq_s \pi_2$ for all $s \in S_{common}(\pi_1, \pi_2)$, then the two plans are equally good, written $\pi_1 \simeq \pi_2$.

Example 5 *Let us assume that the door in the domain of Example 1 can never be "Locked" and, hence, the action "goRight" performed in "Room 3" deterministically moves the robot to "Store". Therefore the plan π_1 defined in Example 3 is a solution for the planning problem with preferences defined in Example 4. We also consider another solution π_2 that is defined as follows:*

state	*action*
$room = Hall$	*goRight*
$room = Room2$	*goDown*
$room = Room1$	*goRight*

These plans have only one common state $s = \{room = Hall\}$. We have $pref(\pi_1, s)^{best} = pref(\pi_1, s)^{worst} = 1$ because all execution paths satisfy g_1. However $pref(\pi_2, s)^{best} = 1$, but $pref(\pi_2, s)^{worst} = 2$ because there is an execution path that satisfies g_2. Therefore $\pi_1 \prec \pi_2$.

Notice that the plans ordering relation is not total: two plans are incomparable if there exist states $s_1, s_2 \in S_{common}(\pi_1, \pi_2)$ such that $\pi_1 \prec_{s_1} \pi_2$ and $\pi_2 \prec_{s_2} \pi_1$. However, in this case we can construct a plan π such that $\pi \prec \pi_1$ and $\pi \prec \pi_2$, as follows:

- if $\langle s, a\rangle \in \pi_1$ and either $s \notin \text{StatesOf}(\pi_2)$ or $\pi_1 \preceq_s \pi_2$, then $\langle s, a\rangle \in \pi$;
- if $\langle s, a\rangle \in \pi_2$ and either $s \notin \text{StatesOf}(\pi_1)$ or $\pi_2 \prec_s \pi_1$, then $\langle s, a\rangle \in \pi$.

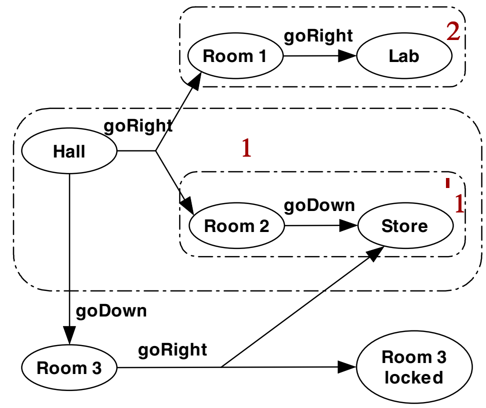


Figure 2: Intuition of the planning algorithm.

As a consequence, there exists a plan which is better or equal to any other plan and we call such plan *optimal*.

Definition 11 (Optimal Plan)
Plan π for problem P is an optimal plan if $\pi \preceq \pi'$ for any another plan π' for the same problem P.

Example 6 *If the door in the domain of Example 1 can be "Locked", then plan π_2 of Example 5 is optimal for the problem defined in Example 4.*

We remark that one could adopt more complex definitions for ordering plans. For instance, one could consider not only the goals with maximum and minimum preferences reachable from a state, but also the intermediate ones. Another possibility would be to compare the quality of plans taking into account the probability to reach goals with a given preference, and/or to assign revenues to the goals in the preference list. This however requires to model probabilities in the action outcomes, while our starting point is to have a qualitative model of the domain and of the goal. With respect to these and other possible models of preferences, our approach has the advantage of being simpler, and the experiments show that it is sufficient in practice to get the expected plans.

Planning Algorithm

We now describe a planning algorithm to solve a planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{list}\rangle$. Our approach consists of building a state-action table π_i for each goal g_i, and then to merge them in a single state-action table π. To build tables π_i, we will follow the same approach exploited in (Cimatti *et al.* 2003) for the case of strong goals without preferences, i.e., we perform a backward search for the plan that guarantees to add a state to a plan only if we are sure that a plan exists from that state. In the context of planning with goal preferences, however, performing a backward search also requires to start from the goal of lowest preferences and to incrementally consider goals of higher preference, as shown in the following example.

```
function BuildPlan(D, gList);
  for (i:=|gList|; i>0; i--) do
    SA := StrongPlan(D, gList[i]);
    oldSA := SA;
    wSt := StatesOf(SA) ∪ gList[i];
    j:=i+1;
    while (j≤|gList|)
      wSt := wSt ∪ StatesOf(pList[j]) ∪ gList[j];
      WeakPreImg := {⟨s,a⟩ : Exec(s,a) ∩ StatesOf(SA)≠ 0};
      StrongPreImg := {⟨s,a⟩ : 0 ≠ Exec(s,a) ⊆ wSt};
      image := StrongPreImg ∩ WeakPreImg;
      SA := SA∪{⟨s,a⟩ ∈ image: s ∉ StatesOf(SA)};
      if (oldSA ≠ SA) then
        SA := SA ∪ StrongPlan(D, StatesOf(SA) ∪ gList[i]);
        oldSA := SA;
        wSt := StatesOf(SA) ∪ gList[i];
        j := i+1;
      else
        j++;
      fi;
    done;
    pList[i] := SA;
  done;
  return pList;
end;
```

```
function StrongPlan(D, g);
  SA := ∅;
  do
    oldSA := SA;
    sSt := StatesOf(SA) ∪ g;
    StrongPreImg := {⟨s,a⟩ : 0 ≠ Exec(s,a) ⊆ sSt};
    SA := SA ∪{⟨s,a⟩ ∈ StrongPreImg: s ∉ StatesOf(SA)};
  while (oldSA ≠ SA);
  return SA;
end;
```

Figure 3: BuildPlan function

Example 7 *Suppose we have the domain presented in Example 1 and the planning problem with preferences of Example 4, i.e., the goal consists of two preferences $\mathcal{G}_{list} = \langle g_1 = \{room = Hall\}, g_2 = \{room = Lab\}\rangle$.*

Our requirement on the plan is that the robot should at least *reach the Lab. For this reason, we start by searching for states from which a strong solution π_2 for goal g_2 exists. We incrementally identify states from which g_2 is reachable with a strong plan of increasing length. This procedure stops when all the states have been reached from which a strong plan for g_2 exists. An example of such a plan π_2 on our robot navigation domain is depicted on the top right part of Figure 2. In this case, the only state from with the Lab can be reached in a strong way is $Room1$.*

We then take goal g_1 into account, and we construct a plan π_1, which is weak for g_1, but which is a strong solution for goal $g_1 \vee g_2$. As in the previous step, we initially build π_1 as a strong solution for the goal g_1, using a backward-search approach. This way, we select those actions that guarantee to reach our preferred goal (see plan π_1' in Figure 2). Once such strong plan for g_1 cannot be further extended, we perform a weakening *of the plan, i.e., we try to find a* weak *state-action pair for π_1 which is, at the same time, a* strong *action-pair for $\pi_1 \cup \pi_2$. Intuitively, if strong planning becomes impossible, we look for a weak extension of the π_1 which leads either to the strong part of the π_1 or to the plan for the less preferable goal π_2. In our example, we add state-action pair $\langle room = Hall, goRight\rangle$ during the* weakening *process. Notice that having already computed π_1 is a necessary condition to apply the weakening. After the weakening, we continue by looking again for strong extensions of plan π_1, until a new weakening is required. Strong planning and weakening are performed cyclically until a fixed point is reached. In our simple example, we reach the fixed point with a single weakening, and the final plan π_1 is shown in Figure 2.*

In case of an arbitrary goal $\mathcal{G}_{list} = \langle g_1, ..., g_n\rangle$, the weakening process for the plan π_i consists of several iterations. We first try apply weakening using $\pi_i \cup \pi_{i+1}$, then $\pi_i \cup \pi_{i+1} \cup \pi_{i+2}$, and so on, until at least one appropriate state-action pair is found.

In the final step of our algorithm we check whether the initial states $\mathcal{I}$ are covered by plans π_1 and π_2 and, if this is the case, we extract the final plan π combining π_1 and π_2 in *a suitable way.*

We now describe the algorithm implementing the idea just described in Example 7. The core of the algorithm is function *BuildPlan*, which is shown on Figure 3.

This function accepts a domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, a list of goals, and returns a list of plans, one plan for each goal. The algorithm builds a plan $pList[i]$ for each goal $gList[i]$ [lines 02-23], starting from the worst one ($i = |gList|$) and moving towards the best one ($i = 1$). For each goal $gList[i]$ we do following steps:

- We first do strong planning for goal $gList[i]$ and store the result in variable SA [line 03]. Function $StrongPlan$ is defined in (Cimatti *et al.* 2003), and we report it for completeness at the end on Figure 3.
- When a fixed point is reached by function $StrongPlan$, a weakening step is performed. Variable $oldSA$ is initialized to SA [line 04] — this is necessary to check whether the weakening process is successful.
- In the weakening [lines 05-21], we incrementally consider goals of lower and lower preference, starting from goal with index $j = i + 1$, until the weakening is successful. Along the iteration, we accumulate in variable wSt the states against which we perform the weakening. Initially, we define wSt as those states for which we have a plan for $gList[i]$, namely, the states in SA and those in $gList[i]$ [line 05]. We then incrementally add to wSt the states for $gList[j]$ [line 08].
- During the weakening process, we look for states-action pairs [line 11] which lead to SA in a weak way [line 09], and that, at the same time, lead to wSt in a strong way [line 10]. We add to SA those state-action pairs that correspond to states not already considered in SA [line 12] — as explained in (Cimatti *et al.* 2003), removing pairs already contained in SA is necessary to avoid loops in the plans.
- If we find at least one new state-action pair [lines 13-18], then we extend SA performing strong planning again [line 14], and re-start weakening, re-initializing $oldSA$, j, and wSt [line 15-17].
- If we reach a fixed point, and we cannot increment SA neither by strong planning nor by the weakening process, then we save SA as plan $pList[i]$ [line 22] and start planning for the goal $gList[i-1]$.

The top level planning function is the following:

```
function  Planning(D, I, gList);
  pList := BuildPlan(D, gList);
  if  I ⊆  ⋃_{1≤i≤|gList|}  (StatesOf(pList[i])∪gList[j])
    π = extractPlan(gList, pList);
     return  π;
   else
     return  ⊥;
  fi ;
end ;
```

Function $Planning$ accepts the planning domain $\mathcal{D}$, the set of domain initial states I, and the goal list $gList$, and returns a plan π, if one exists, or $\perp$. Function $Planning$ checks if the plans computed by $BuildPlan$ for all the goals in list $gList$ are enough to cover the initial states I. If yes, then a plan π is extracted and returned. Otherwise, $\perp$ is returned. Function $extractPlan$ builds a plan by merging the state-action tables in $pList$ in a suitable way. More precisely, it guarantees that a state-action pair from $pList[i]$ is added to π only if this state is not managed by a plan for a "better" goal: $\langle s, a \rangle \in \pi$ only if $\langle s, a \rangle \in pList[i]$ for some i, and $s \notin$ StatesOf$(pList[j]) \cup gList[j]$ for all $j < i$.

The following theorem states the correctness of the proposed algorithm. For lack of space, we omit the proof, which is based on techniques similar to those exploited in (Cimatti *et al.* 2003) for the case of strong planning.

Theorem 1 *Function $Planning(D, I, gList)$ always terminates. Moreover, if $Planning(D, I, gList) = \pi \neq \perp$, then π is an optimal plan for problem $\langle D, I, gList \rangle$, according to Definition 11. Finally, if $Planning(D, I, gList) = \perp$, then planning problem $\langle D, I, gList \rangle$ admits no strong solutions according to Definition 8.*

Experimental Evaluation

We implemented the proposed planning algorithm on the top of the MBP planner (Bertoli *et al.* 2001). MBP allows for exploiting efficient BDD-based symbolic techniques for representing and manipulating sets of states, as required by our planning algorithm.

In order to test the scalability of the proposed technique, we conducted a set of tests in some experimental domains. All experiments have been done by the 1.6GHz Intel Centrino machine with 512MB memory and running a Linux operating system. We consider two domains. The first one is a robot navigation domain, which is defined as follows.

Experimental domain 1 *Consider the domain represented in Figure 4. It consist of N rooms connected by doors and a corridor. Each room contains a box. A robot may move between adjacent rooms if the door between these rooms is not blocked, pick up a box in a room and put down it in the corridor. The robot can carry only one box at the same time. A state of the domain is defined in terms of fluent* `room` *that ranges from 0 to N and describes the robot position, of boolean fluent* `busy` *that describes whether the robot is carrying a box at the moment, of boolean fluents* `door[i][j]`*, that describe whether the door between rooms* `i` *and* `j` *is blocked, and of boolean fluents* `box[i]` *describing whether the box in room* `i` *is on its place in the room. The actions are* `pass-i-j`*,* `pick-up`*,* `put-down`*. Actions* `pass-i-j`*, which changes the robot position from* `i` *to* `j`*, can non-deterministically block* `door[i][j]`*.*

The planning goal expresses different preferences on how boxes are supposed to be delivered to the box storage in the corridor. The most preferable goal is "deliver all boxes". The set of boxes to be delivered is gradually reduced for goals of intermediate preference. The worst goal is "reach the corridor". It means that the plan has to avoid situations where all doors of the room occupied by the robot are blocked. Notice that the door used by the robot to enter the

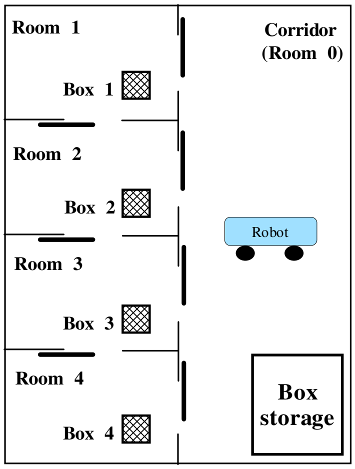


Figure 4: A robot navigation domain

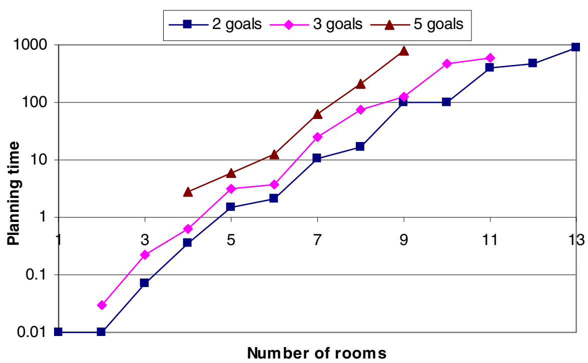


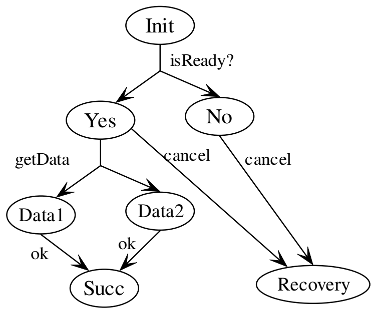
Figure 5: Experiments with robot navigation domain

room can become blocked, so it may be necessary to follow a different path to leave the room, which may lead to more blocked doors and unreachable boxes.

We fixed number of goals and tested the performance of the planning algorithm with respect to the number of rooms in the domain. The results for 2, 3 and 5 goals are shown on the Figure 5. It shows that the proposed technique is able to manage domains of large size: it takes less than 600 seconds to plan for 5 goals in the domain with 9 rooms (i.e., more than 2^{29} states).

The second experimental domain is inspired by a real application, namely *automatic web service composition* (Pistore, Traverso, & Bertoli 2005; Pistore *et al.* 2005). Within the Astro project[1], we are developing tools to support the design and execution of distributed applications obtained by combining existing "services" made available on the web. The planning algorithm described in this paper is successfully exploited in that context to automatically generate the composition of the existing services, given a description of the requirements that the composition should satisfy.

Experimental domain 2 *The planning domain describes a set of "component services", where each service has the following structure:*



The initial action "isReady" forces the component service to non-deterministically decide whether it is able to deliver the requested item (i.e., booking a hotel room or a flight, renting a car, etc.) or not. In the latter case, the only possibility is to cancel the request. If the service is available, it is still possible to cancel the request, but it is also possible to execute the non-deterministic action "getData", which allows us to acquire information on the service (i.e., the name of the booked hotel, or the id for the car rental).

The planning goal expresses different preferences on the task that the web service composition is supposed to deliver. For instance, the most preferred goal g_1 could be "book hotel and flight, and rent a car", the second preference g_2 could be "book hotel and flight without car", and the last preference g_3 could be "book hotel and train" — similar goals are very frequent in the domain of web service composition.

We considered two sets of experiments. In the first set, we tested the performance of the planning algorithm with respect to the size of the planning domain. We considered goals with 1, 7, and 15 preferences, and for all these cases we considered domains with an increasing number of services. The results are shown in the left side of Figure 6. The horizontal axis refers to the number of services composing the domain (notice that n services means 7^n states in the domain). In the vertical axis, we report the planning time in seconds. In the second set of experiments, we test the performance with respect to the size of goals. The results are shown in the right side of Figure 6, where we fixed the size of the domain to 10, 20 and 30 services, and increase the number of preferences in the goal $\mathcal{G}_{list}$ (horizontal axis).

Both sets of experiments show that the algorithm is able to manage domains of large size: planning for 25 goals in a domain composed from 30 services (i.e., more than 2^{82} states) takes about 600 seconds.

Conclusions and Related Work

In this paper we have presented a solution to the problem of conditional planning with goal preferences. We have provided a theoretical framework, as well as the implementation of an algorithm, and we have shown that the approach is promising, since it can deal with large state spaces and complex goal preferences specifications.

[1]The detailed description of the Astro project can be found on `http://www.astroproject.org`.

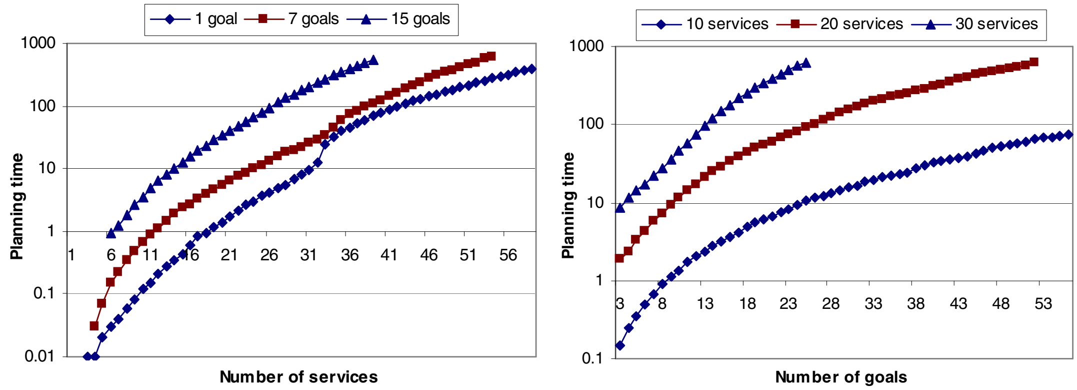


Figure 6: Experiments with service composition domain

Decision theoretic planning, e.g., based on MDP (Boutilier, Dean, & Hanks 1999) is a different approach that can deal with preferences in non-deterministic domains. However, there are several differences with our approach, both conceptual and practical. First, in most of the work on decision theoretic planning goals are not defined explicitly, but as conditions on the planning domain, e.g., as rewards/costs on domain states/actions, while our goal preference model is explicit. Second, we propose a planner that works on a qualitative model of preferences. In decision theoretic planning, optimal plans are generated by maximizing an expected utility function. Finally, from the practical point of view, the expressiveness of the MDP approach is more difficult to be managed in the case of large state spaces.

Apart from planning based on MDP, most of the approaches to planning with goal preferences do not address the problem of conditional planning, but are restricted either to deterministic domains, and/or to the generation of sequential plans. This is the case of planning for multiple criteria (Refanidis & Vlahavas 2003), of the work on over-subscription planning (Briel *et al.* 2004; Smith 2004), of CSP-based planning for qualitative specifications of conditional preferences (Brafman & Chernyavsky 2005), and of preference-based planning in the situation calculus (Bienvenu & McIlraith 2005). In the field of answer set programming, (Eiter *et al.* 2002) addresses the problem of generating sequences of actions that are conformant optimal plans in domains where non-deterministic actions have associated costs. (Son & Pontelli 2004) proposes a language for expressing plan preferences over plan trajectories, whose foundations are similar to those of general rank-based languages for the representation of qualitative preferences (Brewka 2004).

References

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a Model Based Planner. In *Proc. of IJCAI01 Workshop on Planning under Uncertainty and Incomplete Information*.

Bienvenu, M., and McIlraith, S. 2005. Qualitative Dynamical Preferences in the Situation Calculus. In *Multidisciplinary IJCAI'05 Workshop on Advances in Preference Handling*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelliegence Research* 11:1–94.

Brafman, R., and Chernyavsky, Y. 2005. Planning with Goal Preferences and Constraints. In *Proc. ICAPS'05*.

Brafman, R., and Junker, U., eds. 2005. *Multidisciplinary IJCAI-05 Workshop on Advances in Preference Handling*.

Brewka, G. 2004. A rank based description language for qualitative preferences. In *Proc. ECAI'04*.

Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proc. AAAI'04*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence* 147(1-2):35–84.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2002. Answer Set Programming Under Action Costs. *Journal of Artificial Intelligence Research* 19:25–71.

Hoffmann, J., and Brafman, R. 2005. Contigent Planning via Heuristic Forward Search with Implicit Belief States. In *Proc. ICAPS'05*.

Pistore, M.; A.Marconi; Traverso, P.; and Bertoli, P. 2005. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. IJCAI'05*.

Pistore, M.; Traverso, P.; and Bertoli, P. 2005. Automated Composition of Web Services by Planning in Asynchronous Domains. In *Proc. ICAPS'05*.

Refanidis, I., and Vlahavas, I. 2003. Multiobjective heuristic state space planning. *Artificial Intelligence* 145(1-2):1–32.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellegence Research* 10:323–352.

Smith, D. 2004. Choosing Objectives in Over-Subscription Planning. In *Proc. ICAPS'04*.

Son, T., and Pontelli, E. 2004. Planning with Preferences using Logic Programming. In *Proc. LPNMR'04*.