A Planning and Monitoring System for Dynamic Environments

Oscar Sapena, Eva Onaíndia
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Spain
{osapena,onaíndia}@dsic.upv.es

Abstract. SimPlanner is an integrated tool for planning and execution-monitoring which allows to interleave planning and execution. In this paper we present the on-line planner incorporated in SimPlanner. This is a domain-independent planner for STRIPS domains. SimPlanner participated in the IPC 2002, obtaining very competitive results.

Introduction

Off-line planning generates a complete plan before any action starts its execution [17]. This forces to make some assumptions that are not possible in real environments like, for example, that actions are uninterruptable, that their effects are deterministic, that the planner has complete knowledge of the world or that the world only changes through the execution of actions.

On the other hand, on-line planning allows to start execution while the planner continues working in order to improve the overall planning and execution time [17]. Nowadays there are only some few approaches for planning in dynamic environments and/or with incomplete information [10]:

- Conditional planning: there exists two approaches in conditional planning. The first one is based on those problems where the next action to be executed in a plan can depend on the result of previous sensing actions, that is, on information obtained by means of actions during execution time [13]. The second approach tries to consider all the possible contingencies which can happen in the world [4]. Although this solution is untractable in complex environments, it is interesting for particularly dangerous domains. Probabilistic planning is a more moderate variant, since it generates conditional plans only for the most likely problems [6].

- Parallel planning and execution: this approach separates the planning process from the execution [8]. The execution module is able to react to the environment without the necessity of a plan. The planner is in charge of modifying the behavior of this module in order to increase the satisfaction probability of the objectives.

- Interleaving planning and execution: this approach allows quick and effective responses to changes in the environment, and it has been adopted by many researchers [3] [15].
SimPlanner is an integrated tool for planning and execution, and it is based on this latter approach. The on-line planner generates a sequence of actions to reach the goals, while the execution module carries out these actions and provides the planner with sensing information. With this new incorporated information, the planner updates its beliefs about the world.

Objectives

SimPlanner is aimed to be an integrated tool for planning and execution monitoring. SimPlanner has been developed to work under several domains and not only for particular robot environments. Because SimPlanner is thought to be a domain-independent tool, we have chosen PDDL 2.1 [9] as the planning language for domain and problem specification. SimPlanner only uses level one of PDDL (corresponding to the ADL level of the McDermott’s PDDL, without disjunctive preconditions neither conditional effects), although extending it to support levels 2 and 3 (these levels extend level 1 through numeric variables and durative actions) is quite simple.

Our second objective is to design a fast planner so that SimPlanner is able to react rapidly to exogenous events. Moreover, planning should consume less time than the execution, otherwise the behavior of the system could demean and even lose the chance to reach the goals [14]. If this objective is accomplished, the planner will have additional time to optimize the part of the plan that has not been executed yet. Plan quality is not so relevant in dynamic environments as it is not worth spending lots of time in computing a good plan when it may get invalid shortly after execution starts [12].

The objective of this work is to illustrate the working of the on-line planner integrated in SimPlanner. For this reason, the rest of the SimPlanner modules will only be briefly commented in the following section.

SimPlanner overview

The SimPlanner tool is thought to be used in real environments such as the intelligent control of robots. However, it has initially been implemented as a simulator in order to check its behavior without the necessity of integrating it in different several domains. SimPlanner is made up of three components: an on-line planner, a monitoring module and a replanner.

The on-line planner

The on-line planner is responsible for generating, in an incremental way, a plan to achieve the goals. As soon as the planner calculates the first action, the plan can begin to be executed. Starting from this moment, the planning and execution processes continue working in parallel. The on-line planner will be described with more detail in the next section.
The monitoring module

Monitoring is the process of observing the world and trying to find discrepancies between the physical reality and the beliefs of the planner [7]. Contrary to the classic planning, monitoring is needed for different reasons [13]:

- The planner can have an incomplete knowledge of the world in the initial state.
- The effects of the actions can be, sometimes, uncertain.
- Exogenous actions produced by external agents can take place.

There exists mainly two types of plan execution monitoring [10]: action monitoring checks that preconditions are valid before the action execution and that its effects have taken place as expected. The environment monitoring tries to acquire information of the world that can condition the rest of the planning process. Monitoring is, therefore, domain-dependent. Since SimPlanner is being used at the moment as a simulator, this information is input in the system by the user. The user is who decides what information the robot receives and which unexpected events that happen in the world are communicated.

![SimPlanner main interface window](image)

**Fig. 1.** SimPlanner main interface window

Figure 1 shows the graphical interface to monitor a plan execution. In the left upper part of the screen it is shown the literals of the current state of the execution; the lower section shows the literal goals or objectives of the final situation.
On the right side of the screen a graph representing the plan under execution is displayed. The circles stand for the actions in the plan. Those actions ready to be executed at the next time step are double-circled. The right lower part displays information about the action selected by the user in the upper window. At any time during the simulation it is possible to modify the current state to introduce new information from the external world. The user can eliminate those literals which are no longer true or insert new literals in the current state.

The replanner

When an unexpected event is detected, the calculated plan is checked in order to assure that it is still valid [7]. If this is the case, the execution simply continues. Otherwise the replanning module is invoked. The replanner tries to reuse as much of the calculated plan as possible without losing the quality of the final plan. A fully detailed description of the replanning module can be found in [16]. After this step, the on-line planner starts again.

The on-line planner

This work is focus to illustrate the sequential on-line planner integrated in SimPlanner. The planner is based on a depth-first search, with no provision for backup. The reason for not using a complete search algorithm is, once again, the necessity of finding very rapidly the first set of actions to be executed. However, the use of a complete search as iterative deepening could be used at some particular times when there is enough slack time for planning. The planning decisions (inferred actions) are consequently irrevocable in SimPlanner. This approach speeds up the planning process, but presents two shortcomings:

- **Dead-ends**: it is possible to reach a state which it is impossible to achieve the goals from [7].
- **Loops**: in spite of the mechanism SimPlanner uses to detect a previous reached state, the planner can get stuck in a loop which prevents the planner from finding a solution.

Therefore, the planner is not complete, but these shortcomings are acceptable in most of cases due to the advantages it offers against classical off-line planners. Moreover, it is possible to improve the planning performance by taking advantage of the time gained by the planner during the execution.

The overall working scheme is shown in Figure 2. A planning problem \( P = (O, I, G) \) is a triple where \( O \) is the set of actions, \( I \) the initial state and \( G \) the top-level goals. This algorithm starts from the current state \( S_0 \), which initially corresponds to \( I \). The planner calculates the next action to be executed. The current state \( S_0 \) is updated by applying this action. This algorithm will be executed repeatedly until all the goals are achieved (\( G \subseteq S_0 \)).

The SimPlanner planning algorithm can be divided into four main steps:
1. **Non-achieved goals selection:** the non-achieved goals are those which are not true in the current state ($\{g_i : g_i \in G \land g_i \notin S_0\}$).

2. **Calculation of the approximate plans:** an approximate plan is computed for each non-achieved goal $g_i$ separately, i.e., $P$ is decomposed in $m$ planning subproblems $P_1 = (O, S_0, g_1), P_2 = (O, S_0, g_2), \ldots, P_m = (O, S_0, g_m)$.

3. **Grouping of the approximate plans:** approximate plans are grouped according to their initial actions. Each of these groups is called a branch and, all the approximate plans in a branch share, at least, the same first action.

4. **Selection of the action to be executed:** the branches are ordered according to a conflict checking criteria. The next action to be executed will be the first action of the branch ordered in the first place.

We are going to illustrate the algorithm description through an example of the DriverLog domain [2]. This domain involves driving trucks around delivering packages between locations. The difference with respect to a more classical transportation problem such as Logistics [1] is that, in DriverLog, trucks require drivers who must walk between locations in order to reach the truck they must drive. Figure 3 shows the initial state of the problem example. The top-level goals of the problem are: (at package2 s0) -to deliver package2 to the location s0, (at package1 s2) -to deliver package1 to the location s2 and (at driver1 s1) -driver1 must end up in s1 in the goal state.

**Non-achieved goals selection**

The top-level goals of the problem are taken into account during all the planning process, but only those goals that are no currently achieved are considered at each step of the algorithm.

In the example, the non-achieved goals coincide with the top-level goals ((at package2 s0), (at package1 s2) and (at driver1 s1)) because none of the three literals are true in the initial state.
Calculation of the approximate plans

The computation of an approximate plan is incrementally performed in three stages. The starting point is to build a Relaxed Planning Graph (RPG). The second stage generates a special type of graph named Backward Graph (BG) and the final stage is aimed at extracting the approximate plans from the BG.

First stage: RPG

The RPG is a graph based on a GraphPlan-like expansion [5] where delete effects are ignored. The first level of the RPG is a literal level which contains all the literals that are true in the current state $S_i$. The expansion of the RPG finishes when a literal level containing all top-level goals is reached, or when it is not possible to apply any new action. This type of relaxed graph is commonly used in heuristic search based planners [11] as it allows to easily extract admissible heuristics to guide the search.

Second Stage: BG

The BG is a graph whose nodes represent sets of subgoals and whose edges denote clusters of actions. SimPlanner uses a regression process to create a BG for each non-achieved top-level goal $g_i$.

**Definition 1** A cluster for a literal $l_i$ ($C(l_i)$) is the set of actions of the RPG which produce $l_i$: $C(l_i) = \{a_i : a_i \in \text{RPG} \land l_i \in \text{Add}(a_i) \}$

In this regression process, clusters of actions are applied over subgoals, i.e., the application of a cluster $C(l_i)$ to a subgoal set yields a situation in which $l_i$ is achieved [18].

**Definition 2** The application of a cluster $C(l_i)$ to a subgoal set $S$ returns a new subgoal set $S'$ defined as:

$$S' = \text{Result}(C(l_i), S) = S - \text{Add}(C(l_i)) + \text{Prec}(C(l_i)),$$

where

$$\text{Add}(C(l_i)) = \bigcap \text{Add}(a_i), \forall a_i \in C(l_i),$$

and

$$\text{Prec}(C(l_i)) = \bigcap \text{Prec}(a_i) \notin S, \forall a_i \in C(l_i)$$
Definition 3 A BG is defined as a tuple \((N, E)\) where nodes are sets of subgoals and edges represent clusters of actions between two subgoal sets \(S\) and \(S'\). An edge is represented as \(S' \xrightarrow{C(l_i)} S\).

The first level of a BG is formed by a single node, corresponding to the top-level goal \(g_i\). Each node in the BG is expanded by applying clusters of actions to each literal in the node. The BG expansion continues until an empty node is reached. The following algorithm shows, in a more formal way, the BG creation process for a particular goal \(g_i\):

\[
\begin{align*}
N &= \emptyset, \ E = \emptyset \\
n_0 &= \{g_i\} \\
N &= N \cup n_0 \\
End &= \text{false} \\
\text{while} \ -End \ \text{do}: \\
&\quad \quad \quad \quad \quad \quad \ N' = N \\
&\quad \quad \quad \quad \quad \quad \text{for every non-expanded node } n \in N' \ \text{do:} \\
&\quad \quad \quad \quad \quad \quad \quad \quad \ n_{\text{next}} = \text{Result}(C(l), n) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \ \text{if } n_{\text{next}} = \emptyset \ \text{then } End = \text{true} \ \text{endif} \\
&\quad \quad \quad \quad \quad \quad \quad \quad \ N = N \cup n_{\text{next}} \\
&\quad \quad \quad \quad \quad \quad \quad \quad \ E = E \cup n_{\text{next}} \xrightarrow{C(l)} n \\
&\quad \quad \quad \quad \quad \quad \text{endfor} \\
&\quad \quad \quad \quad \quad \quad \text{endwhile} \\
&\quad \quad \quad \quad \quad \quad \text{return } (N, E)
\end{align*}
\]

Figure 4 shows the BG computed for the top-level goal \((\text{at package2 a0})\). The BG starts with a set of subgoals which only contains the top-level goal. Clusters are successively applied to the computed sets until an empty set is reached. When there is only one action in the RPG to achieve a literal, the contents of the cluster is exactly that action. Otherwise, the set of subgoals are composed of the common preconditions and effects of the actions in the cluster plus the literals which are not satisfied yet in the previous subgoal set except those literals which are already true in the current situation. The BGs for the rest of the non-achieved goals ((at package1 a2) and (at driver1 a1)) are computed in the same way.

Third stage: extracting approximate plans

Once the BG is created, our next goal is to select a single action from each cluster. The BG can be viewed as a set of independent sequences of clusters which reversely applied to a top-level goal \(g_i\) lead to an empty set of subgoals.

Definition 4 A path in a BG \(= (N, E)\) (BGpath) is defined as a possible sequence of clusters in the BG. The reverse application of this sequence to a top-level goal leads to an empty set of subgoals:

\[
\text{BGpath} = \{C(l_1), C(l_2), \ldots, C(g_i)\}: \\
S_1 \xrightarrow{C(l_1)} \emptyset, S_2 \xrightarrow{C(l_2)} S_1, \ldots, \{g_i\} \xrightarrow{C(g_i)} S_n \in E
\]
Fig. 4. Backward Graph for the top-level goal (at package2 s0) in the problem example.

In the BG represented in Figure 4, there are three possible sequences of clusters: C3 → C4 → C2 → C1, C4 → C3 → C2 → C1 and C4 → C2 → C3 → C1. So there will be three BGPaths in this BG.

For each BGPath in the BG, SimPlanner creates as many sequences of actions as possible combinations can be formed with the actions in the clusters of the BGPath. Each sequence of actions is called an approximate plan.

Definition 5 An approximate plan (AP) is a possible sequence of actions extracted from a BGPath. Any action in the AP belongs to the cluster located in the same position within the sequence:

\[ AP = \{ a_1, a_2, ..., a_n \} : a_j \in C(l_j) \land C(l_j) \in BGPath \]

Because the number of APs obtained from a BGPath can be very large, SimPlanner applies a heuristic function to select the best-valued approximate plans. This heuristic function uses a conflict-checking procedure to set a value to each possible AP. A conflict \( Conflict(a_j, a_i) \) occurs when an action \( a_j \) ordered before \( a_i \) deletes a precondition of \( a_i \) and there is no intermediate action which restores that literal. The heuristic function \( h \) is incrementally applied while the approximate plans are being computed. Initially, the function is applied over the first action of each AP, following over the first two actions of each AP and so on. The application of \( h \) over a partial AP, which is made up of the first \( i \) actions, is defined as follows:

\[
h(AP_{i,j}) = \begin{cases} 
  i - j, & \text{if } \exists a_j : (a_j < a_i \land Conflict(a_j, a_i)) \land (\exists a_k : a_j < a_k < a_i \land Conflict(a_k, a_i)) \\
  \infty, & \text{if } \forall a_j < a_i, \#Conflict(a_j, a_i)
\end{cases}
\]

The algorithm used to return the best-valued APs is detailed below. Notice that all the APs have the same length \( \text{length}(BGPath) \), i.e. the same number
of actions. This is due to all the APs are obtained from paths in the same BG, which also have the same length.

\[
L = \text{set of all first partial APs} = \{AP_{1,1}\}
\]

\[
\text{for } i = 2 \text{ to length(BGpath)} \text{ do:}
\]

\[
\text{for all } AP_{1,i-1} \in L \text{ do:} \quad // \text{Expansion of each partial AP in } L
\]

\[
L = L - AP_{1,i-1}
\]

\[
AP_{1,i} = AP_{1,i-1} \cup \text{ComputeNextAction}(AP_{1,i-1})
\]

\[
L = L \cup AP_{1,i}
\]

endfor

max_value = \text{max}(h(AP_{1,i}), \forall AP_{1,i} \in L)

L = \{AP_{1,i} : AP_{1,i} \in L \land h(AP_{1,i}) < \text{max_value}\}

endfor

return L

The resulting set \( L \) only contains a small set of all approximate plans obtained from a BG. Additionally, some further criteria are used to reduce even more the number of APs generated for each goal:

- Actions in the plans are reordered, as some plans are sometimes permutations of the same sequence of actions.
- The executability of each AP is checked, inserting additional actions if necessary. Those plans with a smaller number of executable actions are rejected.

Figure 5 shows the computation of the approximate plan for the non-achieved goal (at package2 s0) in the problem example. Although actions (drive truck1 s1 s0 driver1) and (drive truck1 s0 s1 driver1) also belong to the clusters C3 and C4 of the BGPaths, they are not considered as potential first actions in an approximate plan because they are not executable in the current state (since truck1 is currently in location S2). The heuristic function allows to prune the search space and select the best AP, which is represented with bold arrows in the second graph of Figure 5. The APs for the rest of the non-achieved goals are calculated in the same way. These APs are shown in Table 1.

Table 1. Computed APs for each non-achieved goal

<table>
<thead>
<tr>
<th>Non-achieved goal</th>
<th>Approximate plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>at package2 s0</td>
<td>drive truck1 s2 s1 driver1 \rightarrow load package2 truck1 s1 \rightarrow drive truck1 s1 s0 driver1 \rightarrow unload package2 truck1 s0</td>
</tr>
<tr>
<td>at package1 s2</td>
<td>unload package1 truck1 s2</td>
</tr>
<tr>
<td>at driver1 s1</td>
<td>drive truck1 s2 s1 driver1 \rightarrow disembark driver1 truck1 s1</td>
</tr>
</tbody>
</table>
Fig. 5. Approximate plans for each top-level goal in the problem example

Grouping of the approximate plans

Approximate plans are grouped into branches forming a tree topology. All the APs in a branch begin with the same action, which is the root node of the tree. Gradually, APs in a branch diverge to reach their own objectives.

In our problem, the algorithm returns two branches as there are two different initial actions in the calculated APs. Both branches are represented in Figure 6.

Selection of the action to be executed

Branches are ordered in order to find out which branch must be executed in the first place. A branch $B_1$ is ordered before a branch $B_2$ in the following situations:

- **Flexible orders:** let's suppose that the first action of branch $B_1$ produces literal p, and this literal is not deleted throughout the rest of the branch. If the first action of branch $B_2$ needs and also deletes literal p, then branch $B_1$ is ordered before branch $B_2$. This type of situations often occurs in domains like Logistics [1], Zeno-Truck [2], DriverLog [2], etc. Flexible orders are very useful, for example, to order the load and unload actions in transportation domains before moving the involved vehicle.
Fig. 6. Calculated branches in the problem example and a flexible order between them.

- **Non-flexible orders**: Let's suppose that both branches have an action that needs and deletes literal $p$. This is a non-flexible order since it is not possible to order these actions unless an additional action which restores $p$ is inserted between them. If this additional action is found in only one of the branches, then this branch is ordered before the other one. This type of situations often occurs in domains where there exists very strong interactions between the goals, such as *BlocksWorld* [1], *Depots* [2], *FreeCell* [1], etc.

After applying this process, SimPlanner rejects those branches which are not ordered at first place. If there is more than one branch left, some additional criteria are applied in order to select a single branch. For example, if the goals achieved by a branch $B_1$ is a subset of the goals achieved by another branch $B_2$, $B_1$ is discarded. Another rule is to remove those branches with a lower number of executable actions, etc.

In this problem, there are not non-flexible conflicts. However, there exists one flexible conflict due to the second branch needs the truck to remain in location $s2$ and the first branch needs to move the truck away from this location. Therefore, we can execute first the second branch (that is, *unload package1 in location s2*) and, then, start executing the first branch. This flexible order, which selects the second branch as the first one to be executed, is shown in Figure 6.

The first action of the selected branch (*unload package1 truck1 s2* in the example) is inserted at the end of the current plan. The current state is updated through the new action, and all the algorithm is executed again starting from this new current state. The plan generation finishes when all the goals are achieved.

**Results**

SimPlanner has been tested on a wide range of domains: *Blocksworld, Logistics, Monkey*, etc. Moreover, SimPlanner planner participated in the 2002 International Planning Competition (IPC2002). All the data shown in this section are
extracted from the results of this competition (full results of IPC2002 available at http://www.dur.ac.uk/d.plong/competition.html). The most similar planner to SimPlanner in the competition was FF-Speed [11] as it is designed to return sequential plans very quickly. In fact, FF-Speed is probably the fastest planner in its category, at the expense of a loss in the quality of the generated plans.

The graphics (Figures 7 and 8) show a comparative between SimPlanner planner and FF-Speed for the Depots and Satellite domains. Depots domain is a combination of the Logistics and the Blocksworld domains, and the Satellite domain consists in planning and scheduling a set of observation tasks between multiple satellites. An additional serie showing the time that SimPlanner takes to compute the first action to execute (SP 1st action) is also included in these graphics. The times obtained by SimPlanner and FF-Speed are quite similar. FF-Speed is, in general, a bit faster than SimPlanner, although its behaviour is more unpredictable (SimPlanner scales up very well as the size and complexity of the problems increase). But the main contribution of SimPlanner is that it is able to compute the first action of a plan very quickly (only a few tens of seconds in relatively big problems). As the problem resolution is close to the goals, computation time for deducing an action is shorter and shorter. This feature allows the planner to quickly interact in dynamic environments and get the plan adapted to the new situations which can arise (unexpected events, changes in the goals, etc.)

Fig. 7. Time for the Depots domain

With regard to the quality of plans, in general SimPlanner produces longer plans than FF-Speed (Figure 9). The planning approach used by SimPlanner makes difficult to compute high quality plans.

However, since execution usually takes longer than planning, SimPlanner can take advantage of this extra time to optimize the remaining plan. Also it is possible to adjust the heuristics used by the planner to minimize the error rate in
Figure 8. Time for the *Satellite* domain

Figure 9. Quality of the plans for the *Satellite* domain

the action selection process. What we have presented here is still a preliminary version of the on-line planner so it can be improved in many different ways.

**Conclusions and future work**

SimPlanner is a planning tool for working in dynamic environments or with incomplete information. It allows to monitor a plan execution, to recover from changes in the environment and to adapt the plan to the new needs in fractions of a second. The results of the first version of the integrated on-line planner show that it is able to work very efficiently in a wide range of domains.

We are currently extending SimPlanner to handle numeric variables and functions. This is a very important feature in those domains in which distances and consumable resources (batteries, fuel, etc.) are necessary. In these domains, such as intelligent control of robots, it is also important to be able to handle actions with different durations.
On the other hand, we are working on the integration of SimPlanner in a real environment of mobile robots. Aspects that arise in real environments (like what variables to monitor, how to react when the execution is interrupted, etc.) should be handled, producing a more complex and versatile tool.

Acknowledgements

This work has been partially funded by the Spanish Government CICYT project DPI-2001-2094-C03-03 and by the Technical University of Valencia projects UPV-2001-0017 and UPV-2001-0080.

References