Book Title
Book Editors
IOS Press, 2003

An anytime approach for on-line planning

Abstract. In this paper we present a novel planning approach, based on well-known techniques such as goal decomposition and heuristic planning, aimed at working in highly dynamic environments with time constraints. Our contribution is a domain-independent planner to incrementally generate plans under a deliberative framework for reactive domains. The planner follows the anytime principles, i.e a first solution plan can be quickly computed and the quality of the solution is improved as time is available. Moreover, the fast computation of the sequential actions allows the plan to start its execution before it is totally generated, thus giving rise to a highly reactive planning system.

Keywords. On-line planning, Anytime algorithms

1. Introduction

A planner aimed to generate behavior for an agent in complex and dynamic environments, such as computer games or autonomous robots problems, often has to react within a limited period of time. Moreover, the worlds are highly dynamic and unpredictable. In these type of applications, the goal is not to produce optimal plans, but to obtain a response that complies with the environment demands.

There are several planning approaches to deal with dynamic domains. Contingent planning [11], for example, generate plans where some branches are conditionally executed depending on the information obtained during the execution. Another approach is conformant planning [4], which allows to deal with uncertainty on the initial conditions and the action effects without monitoring the plan execution. However, these approaches cannot take into account all possible contingencies and the computation time is often prohibitive. In order to avoid the computational effort of considering all possible unexpected situations during planning time, the on-line planning approaches tackle these situations only when they appear. However, when precomputed behaviors are not available the planner has to react quickly to unexpected events. One of the common techniques to overcome this problem is to follow the anytime paradigm. Anytime algorithms give intelligent systems the capability to trade deliberation time for quality of results [14].

There are two main issues anytime algorithms deal with: interruptibility and quality. Interruptibility implies that the algorithm must be able to be interrupted at any time and provide some answer. The issue of quality implies that the solution is monotonically improved with respect to time. The *CASPER* planner [9], for example, starts with an empty plan and, at each iteration, tries to solve conflicts and achieve new goals. The work discussed in [5] addresses the issue of integrating information about uncertainty into the planning process and also deals with time constraints. Unfortunately, these systems lack the ability to provide a first valid solution within an amount of time. The same

problem occurs in the *PbR* algorithm [1], which assumes there is a polynomial algorithm to compute a valid initial plan for a particular problem. Other approaches, like the hierarchical planner proposed in [3], allows to limit the time for a first solution by means of a domain-dependent rule-based system. *A-UCMP* [7] is another hierarchical anytime planner that requires a library of reactive actions to execute plans containing abstract actions. To sum up, in general either real-time planners use domain-dependent information to have precompiled plans for quick reactions or they cannot provide an initial solution within a time interval.

2. Objectives

The goal of this paper is to present a domain-independent planner able to provide valid actions to an execution agent in environments with time constraints. Our proposal is a deliberative approach, unlike most of the current reactive planners that require a precomputed behaviour, usually implemented as a set of rules, to select an action according to the current world state [12]. This behavior is domain-dependent and takes a lot of time to be computed (or it is manually introduced into the planner).

Our approach is a novel combination of classical planning techniques such as goal decomposition and heuristic planning. This approach shows to be highly competitive when compared to other state-of-the-art planners, in terms of solution quality and time computation. Moreover, our approach follows the principles of the anytime algorithms:

- Interruptibility: time given to infer a single action is limited. Actually, the planner can compute a valid action (according to its current beliefs) in a few milliseconds. Consequently, if necessary, the plan execution can start almost immediately after the planning process has started. This way, we can get rapid reactions when an unexpected event occurs.
- Following the anytime computation, our planner attempts to find a better solution while time is available. This is currently done by artificially limiting the amount of time used to find a solution: the limit is initially set to a few milliseconds and it is successively increased to allow better solutions.

The contribution of this paper is to present and evaluate a novel fast deliberative planner, competitive with other well-known classical planners, and adaptable to reactive and dynamic domains: there is no need of pre-compiled plans, a first solution plan can be computed fast and the solution is incrementally improved according to available time.

3. The planning system

Our planning system is designed to react rapidly to unexpected events. The speed up of this process is achieved by focusing on the most immediate actions to execute rather than searching for a complete plan. An additional reason for this behavior is that plans often become invalid due to frequent changes in the reactive environments.

The working scheme of the planner is quite simple. Given a state and a deadline, the planner searches for an action, executable in that state, which can successfully lead to a goal state. This process is repeated, starting from the resulting state after executing

the last computed action, until a goal state is reached. This scheme is very flexible as it can be used in many different ways. The planning process, for example, can be carried out concurrently (or in an interleaved way) with the execution. This scheme offers many advantages like, for example, that the planner can take into account information which is only available during execution. However, to work concurrently with the execution, the planner must make assumptions about the outcomes of action executions. Following the *assumption-based planning* approach [10], we have considered all actions as deterministic, replanning when an executed action has an unexpected outcome.

Our planner can also be used as an anytime planner, progressively improving a first initial solution while time is available. To obtain this behavior we gradually increase the maximum available time to compute the actions. Both utilization schemes will be discussed in the results section.

4. The planning algorithm

The planning algorithm is based on a greedy action selection technique: the algorithm computes an individual plan for each top-level goal separately, and these plans are then ordered through a conflict checking process in order to select the next action to execute. A planning problem $P = (O, I, G)$ is a triple where O is the set of operators, I the initial state and G the top-level goals. The algorithm starts from the current state S_0, which initially corresponds to I, and works in four stages:

The relaxed planning graph (*RPG*). The *RPG* is a graph based on a *GraphPlan*-like expansion [2] where delete effects are ignored. These type of graphs are commonly used in heuristic planners, like *FF* [8] or *LPG* [6], since they allow to compute good relaxed plans very fast. Our *RPG* includes some additional features such as metric optimization and support for sensing actions [13].

Calculation of the initial plans. In this stage, an incomplete plan is regressively computed for each non-achieved goal $g_i / g_i \in G \wedge g_i \notin S_0$. Therefore, P is decomposed in n planning subproblems $P_1 = (O, S_0, g_1)$, $P_2 = (O, S_0, g_2)$, ..., $P_n = (O, S_0, g_n)$, where n is the number of non-achieved goals.

The process for building an initial plan P_i starts from an empty plan and set of subgoals SG which initially only contains the top-level goal g_i. In each iteration, the most costly (according to the *RPG*) literal, l, is selected from SG, since expanding first the most costly literals usually generates more informed plans. Then, the best-evaluated action, a, which produces l is added to the beginning of P_i. Actions are evaluated according to their cost and the number of conflicts that they cause (literals deleted by the action and required for the next actions in the plan). Finally, the new set of subgoals SG will be formed with the preconditions of a that do not hold in S_0. This process continues until SG becomes empty.

An initial plan for a top-level goal is not necessarily executable since some of the actions in the sequence might not be applicable in their corresponding state. This is because the algorithm only takes into account one subgoal in each iteration. However, the objective of initial plans is accomplished: an incomplete plan is rapidly computed, the first action is directly executable, and it can be used as a good starting point for further refinements.

The refinement stage. Once P_i for each top-level goal g_i is computed, the refinement stage begins. Plans are improved while there is available time. If a plan P_i is not valid, then there is (at least) one action in P_i with unsupported preconditions. In each iteration, an unsupported precondition is selected and repaired in order to achieve a more complete plan.

In order to repair a precondition p of an action a, a number of (incomplete) plans to achieve p are computed, in the same way than the initial plans. These new plans start from the states prior to the problem (states in P_i previous to action a). Then, one of these plans is selected in order to repair p. The selected plan is the one that produces a fewer number of conflicts (unsupported preconditions). When two plans produce the same number of conflicts, the one with the lowest cost (with regard to the problem metric) is selected.

Selection of the action to be executed. At this point, we have a plan P_i, which might not be totally executable, for each top-level goal g_i. When the executor requests the planner one action, the refinement stage is halted. The planner will return a_{next}, the first action of one or more plans (in case several plans share the same first action). In order to find out which plan must be executed in first place, we apply some criteria to rule out plans. Let's suppose a_{i0} is the first action of a plan P_i and a_{j0} is the first action of another plan P_j:

- **Inverse actions:** P_i is ruled out when a_{i0} is an inverse of the last executed action.
- **Shared action sequences:** let's consider that $P_i = \{a_{i0}, a_{i1}, \ldots, a_{in}\}$ and $P_j = \{a_{j0}, a_{j1}, \ldots, a_{jm}\}$. P_i is ruled out if P_j includes the first action of plan P_i ($a_{jk} = a_{i0}$), and the sequence of actions $\{a_{j0}, \ldots, a_{jk}, a_{i1}, \ldots, a_{in}\}$ is executable. In other words, plan P_j can start its execution without affecting the later execution of P_i.
- **Non-flexible conflicts:** let's suppose that both a_{i0} and a_{j0} need and delete literal l. This is a non-flexible order since it is not possible to order these actions unless an additional action which repairs l is inserted between them. If this additional action is only found in one of the plans (i.e. P_j), then P_j is ordered before P_i and, therefore, P_i is ruled out. This type of situations often occur in domains with strong interactions between goals.
- **First action replacements:** if a_{j0} can be replaced by a_{i0} without causing conflicts, then P_j is ruled out.
- **Flexible conflicts:** let's suppose that a_{i0} requires literal l, and l is not deleted throughout the rest of the plan P_i. If a_{j0} needs and also deletes l, then P_j is rejected. This is because a_{j0} cannot be ordered before a_{i0}. Flexible orders are very useful, for example, to order a *load* and an *unload* action in transportation domains before moving the involved vehicle.

If the result of this filtering process is still a set of plans with different initial actions, then some additional criteria are applied. The first action a_{next} of the selected plan is sent to the executor, and the planner updates its environment model with the expected effects of a_{next}.

5. Results

First, we will consider the on-line approach, where the planner provides the executor actions within a given deadline. This behavior can be observed in Figure 1, correspond-

ing to a problem in the *Rovers* domain and a problem in the *ZenoTravel* domain. These problems can be found in the *IPC-3* [1]. These results have been obtained using a 2 *Ghz. Pentium IV* computer with 512 *Mb.* of memory. Although the data in these figures correspond to a particular problem instance, the behavior is very similar to other instances and domains. As it can be observed, the first actions in the plan are, in general, harder to compute since the distance to the goals is greater. On the contrary, when computing the last actions, the number of goals to reach may be very small since most of them have already been achieved. Consequently, the plans for each goal are shorter and need fewer refinement steps to be repaired.

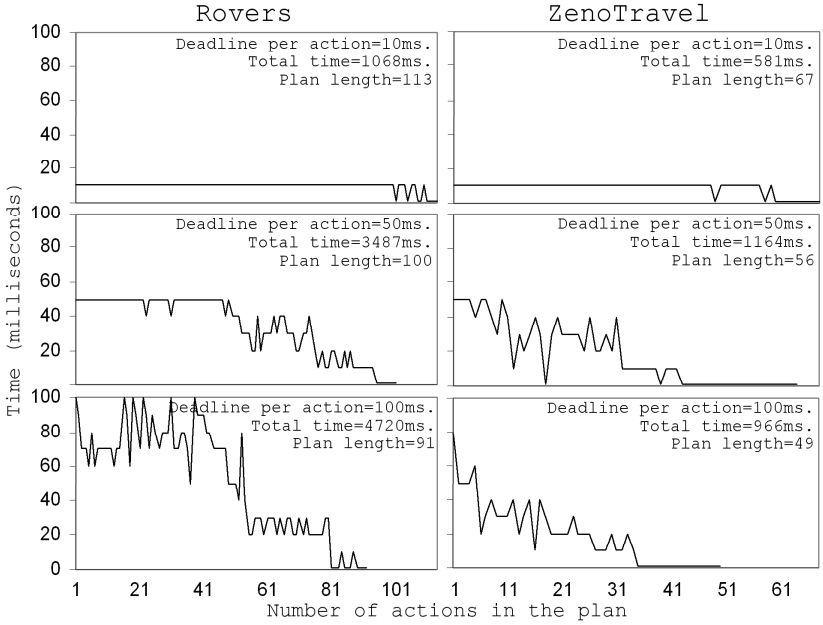

Figure 1. Plans computed for a problem in the *Rovers* (in the left) and a problem in the *ZenoTravel* domain (on the right), progressively increasing the maximum available time to compute each action.

In the examples of Figure 1, the deadline is set to 10, 50 and 100 ms. per action. For the case of 100 ms. deadline, the planner makes a decision before 100 ms. (except for some of the first actions). In the other two cases, the refinement process is interrupted during the computation of many plan actions because the deadline is exhausted and, consequently, worse decisions are taken. In general, the more time available for computation, the better plans are obtained. The anytime behaviour is based on this idea, that is, artificially increasing the available time to compute each action. In our planner, we have used deadlines of 5, 10, 15, 20, 50, 100, 200, 500 and 1000 ms. per action. With greater deadlines, the planner does not improve the quality of the solution since the refinement stage usually terminates before exhausting the available time.

However, our approach presents an inconvenient at this point. The refinement stage stops when the algorithm cannot further improve the current solution, even if it has available time. Yet, the results show that the overall plan quality is good enough when com-

[1] Third International Planning Competition. (*http://planning.cis.strath.ac.uk/competition/*)

Table 1. Solutions found in 20 seconds per plan for several problems in the *Blocksworld* domain. The problem size represents the number of blocks in the problem.

Problem	Our solutions (plan length/time sec.)	*LPG* solutions (plan length/time sec.)
10	64/0.03, 34/0.06	100/0.28, 36/0.5, 34/2.28
11	128/0.05, 32/0.12	200/0.7, 40/1.15, 38/1,9, 36/4,1, 34/5.4, 32/6,52
12	56/0.03, 42/0.12, 38/0.21	154/0.59, 48/1.03, 44/1.21, 42/2.92, 40/3.98
13	72/0.04, 48/0.12, 42/0.19	174/3.06, 52/3.44, 44/4.67, 42/10.44
14	86/0.06, 44/0.18	130/0.77, 64/2.4, 48/2.6, 46/3.4, 44/3.55, 42/5.4
15	100/0.08, 52/0.18, 48/0.31	172/6.23, 50/13.41, 46/13.57
16	188/0.2, 80/0.53, 58/0.72, 54/1.3	192/14.16, 112/16.14, 60/17.27, 58/17.83
17	236/0.25, 68/0.47, 54/0.66	352/15.63, 62/19.9
18	310/0.35, 70/0.6, 62/1.7	-
19	128/0.18, 94/0.6, 86/1.3, 68/2.6, 64/4.4	-
20	106/0.16, 78/0.5, 66/2.7	-

Table 2. Solutions found in 20 seconds per plan for several problems in the *Satellite* domain (*IPC-3*).

Problem	Our solutions (plan length/time)	*LPG* solutions (plan length/time sec.)
10	32/0.07, 31/0.47	32/0.09, 31/0.23, 29/4.48
11	35/0.09, 34/0.34	35/0.1, 34/0.16, 33/1.07, 31/1.7
12	43/0.15	51/0.21, 45/0.33, 43/0.5
13	58/0.29	67/0.36, 65/0.96, 59/2.11, 58/2.28, 57/2.46
14	45/0.17, 44/0.78	45/0.23, 43/0.62, 42/0.76, 41/7.23, 40/12.55
15	51/0.27, 50/1.22	74/0.34, 71/0.52, 64/0.62, 57/1.09, 51/1.31, 50/13.97
16	52/0.34, 49/1.2	54/0.33, 53/0.57, 51/1.31, 50/6.42
17	47/0.34, 46/1	55/0.43, 54/0.54, 53/0.86, 49/1.43, 47/5.32, 46/5.94
18	36/0.12	43/0.2, 41/0.32, 35/0.4, 33/9.88, 32/10.51
19	71/0.34, 65/1.3, 63/2.3	76/0.32, 73/0.54, 72/1.1, 69/1.45, 68/1.6, 64/8.6
20	89/0.5	108/0.4, 105/0.7, 102/1, 101/2, 99/3.4, 94/4, 88/16

pared to other state-of-the-art planners (see Tables 1, 2 and 3). We are currently addressing this problem in a future extension of this work.

We have chosen *LPG v1.2* [6] to test and compare the anytime behaviour of our planner. The reason is that *LPG* is able to provide an initial solution very rapidly and then improve such a solution progressively. Tables 1, 2 and 3 show the successive solutions that our planner and *LPG* have generated for the *Blocksworld*, *Satellite* and *Numeric Depots* respectively. *Numeric Depots* is the numeric version of the *Depots* domain, where some objects must be transported with trucks and arranged in stacks. In this domain, plan quality is not measured in number of actions but in units of consumed fuel.

Results show that our planner is able to compute a first plan more rapidly than *LPG*. We must also take into account that, in our case, it is not necessary to wait for the complete plan to be computed to start its execution. Moreover, our planner scales better than *LPG* in the presented domains. This can be also seen in Figure 2, where we show a com-

Table 3. Solutions found in 30 seconds per plan for several problems in the *Numeric Depots* domain (*IPC-3*).

Problem	Our solutions (fuel-cost/time sec.)	*LPG* solutions (fuel-cost/time sec.)
10	66/0.1	48/0.12, 47/0.18
11	160/0.68, 140/4.4, 120/7.9	159/4, 154/7, 134/9, 133/11, 123/21, 103/28
12	227/2.68, 125/5.28	288/9.1
13	56/0.08	107/0.16, 77/0.21, 68/0.33, 57/0.87
14	89/0.33, 78/1.51	99/1.92, 80/2.19, 78/2.96, 68/3.63
15	256/2.92, 134/13.16	204/28.59
16	57/0.23	79/0.33, 78/0.66, 68/0.79, 59/0.96, 58/1.09, 57/6.4
17	46/0.3	49/0.53, 48/1.44, 38/1.6
18	409/3.11, 113/7.9, 99/12.62, 86/29.42	167/5.17, 137/8.75
19	98/0.5	218/0.9, 201/1.4, 181/1.7, 180/2, 150/2.4, 100/5.6
20	226/6.24, 213/28.99	245/26.65

parison between the time needed by our planner and *LPG* to compute a first solution in the *ZenoTravel* domain. This domain, presented in the *IPC-3*, is a transportation domain, where objects are transported via aeroplanes.

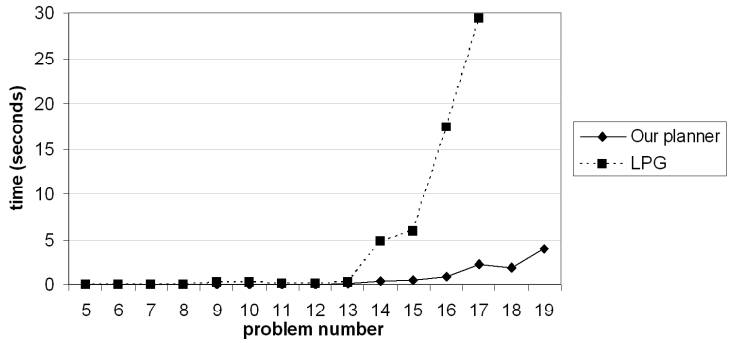

Figure 2. Time to compute a first solution for our planner and *LPG* planner in the *ZenoTravel* domain.

As for the plan quality, the first solutions of our planner are usually better than the ones of *LPG* and, in general, our planner finds equal or better quality solutions in less time than *LPG*.

6. Conclusions and future work

In this paper we have presented an anytime deliberative planner, designed to work in highly dynamic environments with time constraints. This planner is based on well-known techniques, such as goal decomposition and heuristic planning, but combined in a novel approach. Used as an on-line planner, our approach presents important advantages: our planner can obtain a first solution plan very rapidly and scales up very well in many domains. Also, plan execution can start after the first action has been computed, which

can be done in a few milliseconds. This feature allows our planner fast reactions when unexpected events occur.

Our planner can also be used as an anytime planner by artificially increasing the available time to compute each action. Results show that plans obtained are quickly computed and have good quality. However, our planner is not complete, and there is a limit in the quality of the plans it can generate. Currently, we are interested in overcoming this limitation by including an additional search process when there is available time. This way, we can obtain a complete planner, able to produce better-quality plans.

References

[1] Ambite, J., and Knoblock, C.: Planning by rewriting, JAIR **15** (2001), 207–261.
[2] Blum, A., Furst, M.: Fast planning through planning graph analysis, Artificial Intelligence **90** (1997), 281–300.
[3] Briggs, W., Cook, D.: Anytime planning for optimal tradeoff between deliberative and reactive planning, FLAIRS Conference (1999), 367–370.
[4] Bryce, D., Kambhampati, S.: Heuristic guidance measures for conformant planning, ICAPS (2004), 365–375.
[5] Dean, T., Kaelbling, L., Kirman, J., Nicholson, A.: Planning under time constraints in stochastic domains, Artificial Intelligence **76** (1995), 35–74.
[6] Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in LPG, JAIR **20** (2003), 239–290.
[7] Hawes, N.: Anytime planning for agent behaviour, PLANSIG (2003), 157–166.
[8] Hoffman, J., Nebel, B.: The FF planning system: Fast planning generation through heuristic search, JAIR **14** (2001), 253–302.
[9] Knight, R., Rabideau, G., Chien, S., Engelhardt, B., Sherwood, R.: Casper: Space exploration through continuous planning, IEEE Intelligent Systems **16** (2001), 70–75.
[10] Koenig, S., Tovey, C., Smirnov, Y.: Performance bounds for planning in unknown terrain, Artificial Intelligence **147** (2003), 253–279.
[11] Majercik, S., Littman, M.: Contingent planning under uncertainty via stochastic satisfiability, Artificial Intelligence **147** (2003), 119–162.
[12] Pryor, L., Collins, G.: Planning for contingencies: A decision-based approach, JAIR **4** (1996), 287–339.
[13] O. Sapena, E. Onaindía.: Handling numeric criteria in relaxed planning graphs, LNAI **3315** (2004), 114–123.
[14] Zilberstein, S.: Using anytime algorithms in intelligent systems, AI Magazine **17** (1996), 73–83.