An on-line approach for planning in time-limited situations

Oscar Sapena, Eva Onaindía

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Spain
{osapena,onaindia}@dsic.upv.es

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.

1 Introduction

A planner aimed to generate behavior for an agent in complex dynamic and unpredictable environments, such as computer games or autonomous robots problems, often has to react within a limited period of time. 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 [13], 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 [16].

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 [10], for example, starts with an

empty plan and, at each iteration, tries to solve conflicts and to achieve new goals. The work discussed in [6] 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* [8] 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 pre-computed behavior, usually implemented as a set of rules, to select an action according to the current world state [14]. This domain-dependent behavior is very costly 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 increasing 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 quickly computed 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 dynamic 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 and 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, working concurrently with the execution, implies the planner must make assumptions about the outcomes of action executions. Following the *assumption-based planning* approach [12], 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. Since there is no a systematic search process to explore all possible alternatives, there is a limit in the quality of the plans we can obtain. It would be feasible to incorporate this search process into the planner, but this improvement will be addressed in future works. Both utilization schemes, the on-line and the anytime, will be discussed in section 5.

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* [9] or *LPG* [7], since they allow to compute accurate relaxed plans very fast. Our *RPG* includes some additional features such as metric optimization and support for sensing actions [15].

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 the 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, as expanding firstly 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 - which we call a_{fail} - in P_i with unsupported preconditions. In each iteration, an unsupported precondition, p, of a_{fail} is selected and repaired in order to achieve a more complete plan.

In order to repair precondition p of a_{fail}, a number of (incomplete) plans to achieve p are computed, in the same way that the initial plans. These new plans start from the states in P_i - which we call S_j - previous to action a_{fail} and lead to states S'_j in which p is true. These plans can be observed as bold arrows in Figure 1. In order to reuse the final part of plan P_i, from each state S'_j we compute new (incomplete) plans to reach an existing state in P_i, taking care not to delete literal p. The new plans (see dashed arrows in Figure 1) are also computed as initial plans.

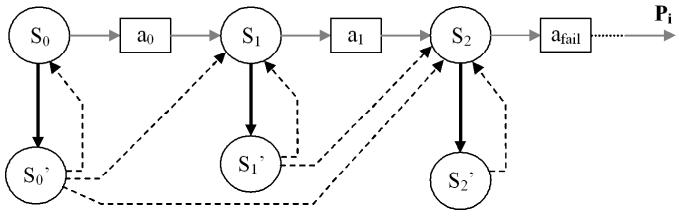

Fig. 1. Different alternatives to repair a precondition of a_{fail}.

The plans computed in this refinement stage provide different alternatives to repair precondition p. These alternatives can be observed in Figure 1 by following the arrows (plans) between the states. From all possible alternatives, the one that produces fewer conflicts (unsupported preconditions) is selected. When two alternatives 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. These criteria are classified into:

– **Context criteria:** These criteria study the last executed actions in order to rule out those plans that do not make progress toward the goals, that is, plans that undo a previously achieved subgoal (inverse actions) or plans that cause an execution loop.
– ***Mutex* criteria:** These criteria study the *mutex* relations [2] between actions of different plans P_i. These relations are used to establish an order among plans in such a way that those plans that do not result ordered in first place are ruled out.
– **Least commitment criteria:** If a plan P_i can start its execution without affecting the later execution of other plan P_j, then P_j is ruled out. This way, we postpone the decision to later stages.

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 show the obtained results from the perspective of the anytime behavior of our planner. We have chosen *LPG v1.2* [7] to test and compare our results because *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 (these results have been obtained using a 2 *Ghz. Pentium IV* computer with 512 *Mb.* of memory). *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. These problems can be found in the *IPC-3* (Third International Planning Competition, *http://planning.cis.strath.ac.uk/competition/*).

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 comparison 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. 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*.

Now, we will show the on-line behavior of our planner through the simulation of unexpected events and changes in the goals during the plan execution. For this purpose, we have defined a new robot manipulation domain in which a robot arm is used to pick up objects from some tables and to arrange them in a two-dimensional space around a central piece. The first proposed problem (see Figure 3) consists of ten objects (`p1`, . . ., `p10`) stacked on five tables (`table1`, . . ., `table5`) which must be arranged around a fixed central object `C` placed in a separated platform. An object can be assembled to the left, right, top or bottom of another object which has been already assembled.

Table 1. Solutions found in 20 seconds per plan for several problems in the *Blocksworld* domain.

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.

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

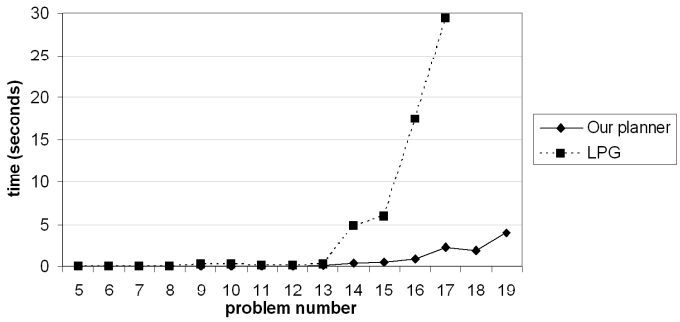

Fig. 2. Time to compute a first solution for our planner and *LPG* in the *ZenoTravel* domain.

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

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, 68/0.8, 59/0.96, 58/1.1, 57/6.4
17	46/0.3	49/0.53, 48/1.44, 38/1.6
18	409/3.1, 113/7.9, 99/12.6, 86/29.4	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

This problem is currently a challenge for many domain-independent planners: *LPG v1.2*, *SGPlan* [5] and *Metric FF* [9] take 2 minutes, 5 minutes and more than an hour respectively to find a solution (without considering unexpected events or goal changes because of their off-line nature).

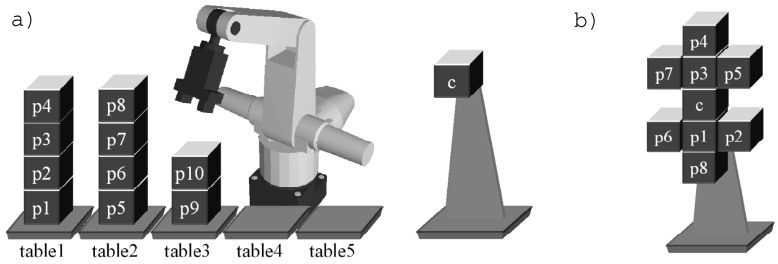

Fig. 3. a) Initial state and b) goal state for the first proposed problem.

For this problem, we have simulated two unexpected events during the execution (see Figure 4). The first unexpected event occurs when the robot has assembled `p1`, `p3` and `p8`, and it is holding `p7` (Figure 4a). At that moment, the robot detects that `p7` has fallen onto table `table4`. The second unexpected event occurs when the robot tries to assemble `p5` to the right of `p3`. An error causes that `p5` is assembled to the right of `p2`. The robot arm can disassemble objects, so it must repair this error.

Figure 5 shows the computation time for each action in the plan. We have limited the available time to one second per action in order to obtain a smooth execution. This limitation together with the problem complexity causes the execution of some unnecessary actions. Firstly, it can be observed that the first actions in the plan are, in general, harder to compute since the distance to the goals is greater. On the contrary, when com-

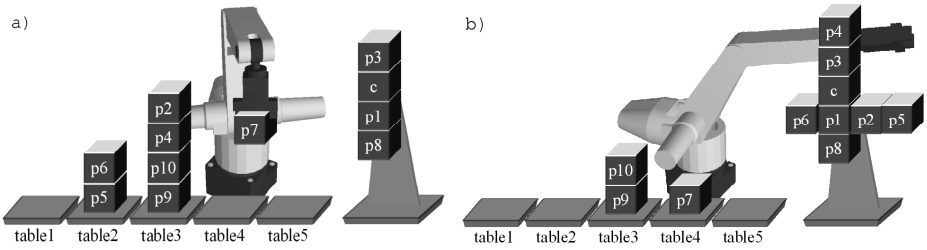

Fig. 4. a) First unexpected event: object $p7$ falls on $table4$. b) Second unexpected event: $p5$ is assembled to the right of $p2$ instead of to the right of $p3$.

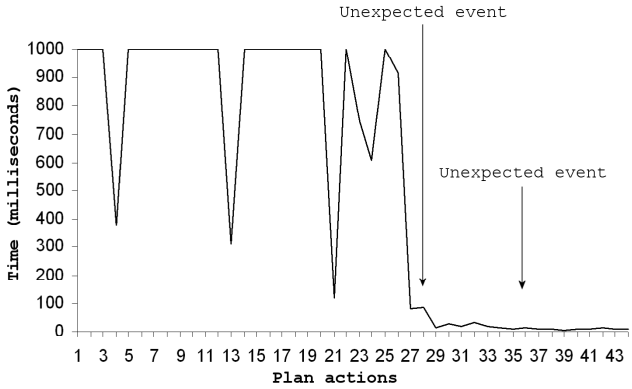

Fig. 5. Computation time for each plan action for the problem with unexpected events.

puting the last actions, the number of non-achieved goals may be very small since most of them have already been accomplished. Consequently, the plans for each goal are shorter and need fewer refinement steps to be repaired.

It can also be observed that an unexpected event does not cause an appreciable increase in the computation time. This is due to the fact that the planner does not reuse previous calculations in order to compute an action. This way, the cost of computing the next action when everything goes as expected is very similar to the cost of computing a first action for a new situation.

In Figure 6 we show a second problem in order to illustrate the planner behavior when faced with changes in the goals. The top of this figure shows the current state of the assembly platform when the change in the goals occurs, whereas the bottom shows the new goal state to be achieved. Figure 7 shows the computation time to calculate each plan action and the instants at which a goal change occurs. It can be observed that a goal change has a similar effect to an unexpected event, that is, a negligible increase in the computation time. Unlike other replanning tools, the behavior of our system does not depend on how drastic the change is, but on the distance from the current state to the goals (actually, many replanners rely on the idea that the actual situation is only slightly different from the original one [11]).

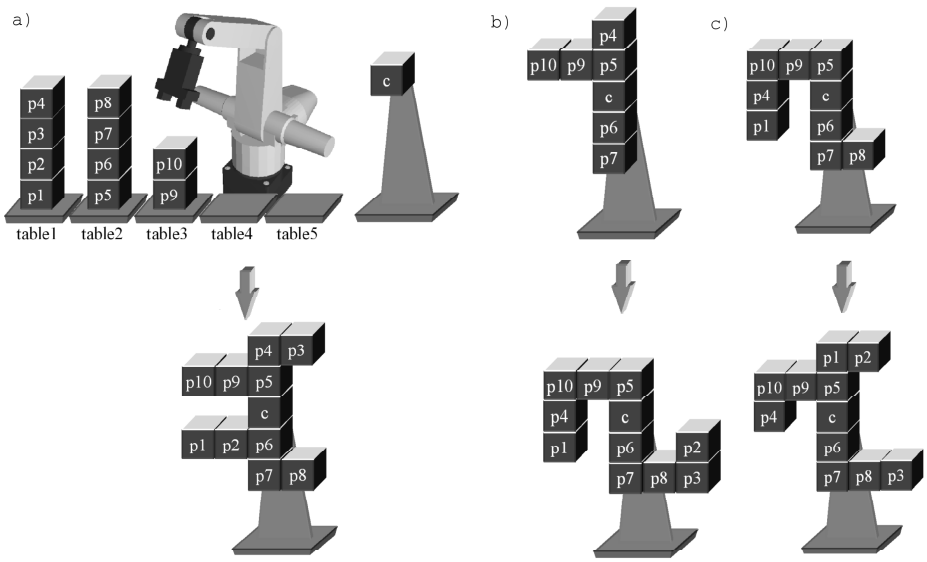

Fig. 6. a) Initial state (top) and goal state (bottom) for the second proposed problem. b) Current state (top) when the first change in the goals (bottom) occurs. c) Second change in the goals.

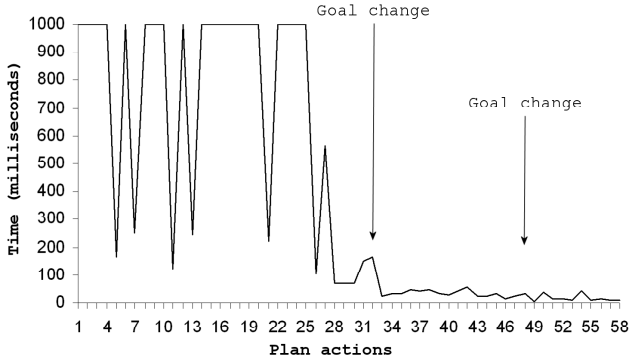

Fig. 7. Computation time for each plan action for the problem with goal changes.

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.

Acknowledgements

This work has been partially supported by the *MCyT* TIN2005-08945-C06-06 (*FEDER*) project.

References

1. J.L. Ambite and C.A. Knoblock, 'Planning by rewriting', *JAIR*, **15**, 207–261, (2001).
2. A. Blum and M. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**, 281–300, (1997).
3. W. Briggs and D.J. Cook, 'Anytime planning for optimal tradeoff between deliberative and reactive planning', *FLAIRS Conference*, 367–370, (1999).
4. D. Bryce and S. Kambhampati, 'Heuristic guidance measures for conformant planning', *Proceedings of ICAPS*, 365–375, (2004).
5. Y. Chen, C.W. Hsu, and B.W. Wah, 'SGPlan: Subgoal partitioning and resolution in planning', *IPC-4 Booklet (ICAPS)*, (2004).
6. T. Dean, L.P. Kaelbling, J. Kirman, and A. Nicholson, 'Planning under time constraints in stochastic domains', *Artificial Intelligence*, **76**(1-2), 35–74, (1995).
7. A. Gerevini, A. Saetti, and I. Serina, 'Planning through stochastic local search and temporal action graphs in LPG', *JAIR*, **20**, 239–290, (2003).
8. N. Hawes, 'Anytime planning for agent behaviour', *PLANSIG*, 157–166, (2001).
9. J. Hoffman and B. Nebel, 'The FF planning system: Fast planning generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).
10. R. Knight, G. Rabideau, S.A. Chien, B. Engelhardt, and R. Sherwood, 'Casper: Space exploration through continuous planning', *IEEE Intelligent Systems*, **16**(5), 70–75, (2001).
11. S. Koenig, D. Furcy, and C. Bauer, 'Heuristic search-based replanning', *International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 310–317, (2002).
12. S. Koenig, C.A. Tovey, and Y.V. Smirnov, 'Performance bounds for planning in unknown terrain', *Artificial Intelligence*, **147**(1-2), 253–279, (2003).
13. S.M. Majercik and M.L. Littman, 'Contingent planning under uncertainty via stochastic satisfiability', *Artificial Intelligence*, **147**(1-2), 119–162, (2003).
14. L. Pryor and G. Collins, 'Planning for contingencies: A decision-based approach', *Journal of Artificial Intelligence Research*, **4**, 287–339, (1996).
15. O. Sapena and E. Onaindía, 'Handling numeric criteria in relaxed planning graphs', *Proceedings of Iberamia (LNCS)*, **3315**, 114–123, (2004).
16. S. Zilberstein, 'Using anytime algorithms in intelligent systems', *AI Magazine*, **17**(3), 73–83, (1996).