Domain-independent On-line Planning for STRIPS Domains

Oscar Sapena, Eva Onaindía

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Spain
{osapena,onaindia}@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 [18]. 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 the execution to begin while the planner is still planning, to improve the combined planning and execution time [18]. Nowadays there are only some few approaches for planning in dynamic environments and/or with incomplete information [9]:

- *Conditional planning*: there exists two approaches in conditional planning. The first one is based in 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 [2]. 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 [4] [6]. The appearance of unpredictable or unlikely situations is usually handled by means of replanning algorithms.
- *Parallel planning and execution*: this approach separates the planning process from the execution [7]. 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 [1] [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 [8] as the planning language for domain and problem specification. SimPlanner only uses level one of PDDL [1] (without disjuntive preconditions neither conditional effects), although extending it to support levels two and three[2] is quite simple.

Our second objective is to design a fast planner so that SimPlanner was able to react rapidly to exogenous events. Also, planning should spend less time than the execution since, otherwise, the behavior of the system could demean and even to 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. 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 as, for example, the intelligent control of robots. However, it has been implemented initially as a simulator in order to check its behavior without the necessity of integrating it in different particular domains.

The on-line planner takes charge of 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 the execution continue working in parallel.

Monitoring is the process of the world observation, trying to find discrepancies between the physical reality and the beliefs of the planner [5]. 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.

[1] Corresponding to the ADL level of the McDermott's PDDL.
[2] Levels 2 and 3 extend level 1 through numeric variables and durative actions.

– Exogenous actions produced by external agents or by the nature can take place.

There exists mainly two types of execution monitoring of a plan [9]: *action monitoring* checks that preconditions are valid before the action execution and 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 introduced in the system by the user. The user is the one that decides what information the robot receives and which are the unexpected events that happen in the world.

When an unexpected event is detected, the calculated plan is checked in order to assure that it is still valid [5]. If this is the case, the execution simply continues. Otherwise the replanning module is called [16]. The replanner tries to take advantage of most of the calculated plan without losing the quality of the final plan. After this step, the on-line planner starts again.

The on-line planner

This work is focus to illustrate the on-line planner integrated in SimPlanner. SimPlanner planner is an incremental sequential planner [17]. It is based on a depth-first search, with no provision for backup. The planning decisions (inferred actions) are consequently irrevocable. 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 [5].
– *Loops*: in spite of the mechanisms for detecting loops, the planner can get stuck in a loop which prevents the planner from finding a solution.

Therefore, the planner is not complete, but this shortcomings are acceptable in lots of cases due to the advantages it offers against classical off-line planners. Moreover, it is possible to improve the planning success rate by taking advantage of the time won by the planner during the execution, in order to optimize the final plan.

The overall working scheme is shown in Figure 1. A planning problem $P = (O, I, G)$ is a triple where O si 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 through applying this action. This algorithm will be executed repeatedly until all the goals are achieved ($G \subset S_0$).

The SimPlanner planner 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 ($\{g_i : g_i \in G \wedge g_i \notin S_0\}$).
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 $P_1 = (O, S_0, g_1)$, $P_2 = (O, S_0, g_2)$, ..., $P_m = (O, S_0, g_m)$.

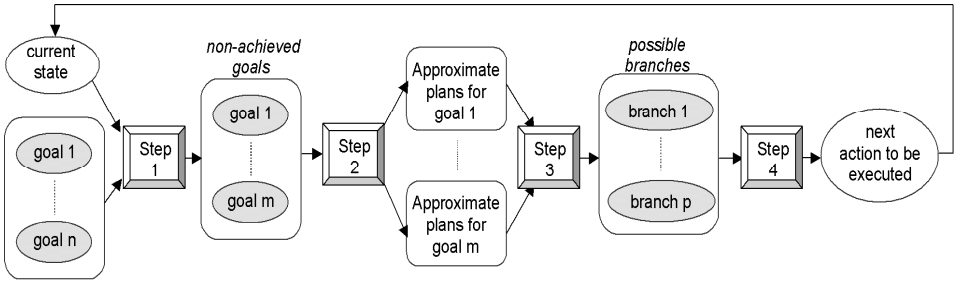

Fig. 1. SimPlanner planner working scheme

3. *Grouping of the approximate plans*: approximate plans are grouped according to their initial actions. Each of this 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.

Steps two and four are the most complex tasks, so they are fully detailed in the next sections.

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 [3] 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_0. 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 [10] [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:*

$C(l_i) = \{a_i : a_i \in RPG \wedge l_i \in Add(a_i)\}$

In this regression process, clusters of actions are applied over subgoals, i.e., the *application* of a cluster $C(l_i)$ in a subgoal set yields a situation in which l_i is achieved [19].

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

$S' = Result(C(l_i), S) = S - Add(C(l_i)) + Prec(C(l_i))$, *where*

$Add(C(l_i)) = \cap Add(a_i), \forall a_i \in C(l_i)$, *and*

$Prec(C(l_i)) = \cap Prec(a_i) \notin S_0, \forall a_i \in C(l_i)$

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 $S' \stackrel{C(l_i)}{\longrightarrow} S$.*

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:

```
N = ∅, E = ∅
n_0 = {g_i}
N = N ∪ n_0
End = false
while ¬End do:
    N' = N
    for every non-expanded node n ∈ N' do:
        for every subgoal l ∈ n do:
            n_new = Result(C(l), n)
            if n_new = ∅ then End = true endif
            N = N ∪ n_new
            E = E ∪ n_new --C(l)--> n
        endfor
    endfor
endwhile
return (N, E)
```

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:*

$BGpath = \{C(l_1), C(l_2), ..., C(g_i)\}$:

$$S_1 \stackrel{C(l_1)}{\longrightarrow} \emptyset, S_2 \stackrel{C(l_2)}{\longrightarrow} S_1, ..., \{g_i\} \stackrel{C(g_i)}{\longrightarrow} S_n \in E$$

For each BGpath in a 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:*
$AP = \{a_1, a_2, ..., a_n\} : a_j \in C(l_j) \wedge C(l_j) \in BGpath$

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:

$$h(AP_{1..i}) = \begin{cases} i - j, & \text{if } \exists a_j : (a_j < a_i \wedge Conflict(a_j, a_i)) \wedge \\ & \quad (\nexists a_k : a_j < a_k < a_i \wedge Conflict(a_k, a_i)) \\ \infty, & \text{if } \forall a_j < a_i,\ \nexists Conflict(a_j, a_i) \end{cases}$$

The algorithm used to return the best-valued APs is detailed below. Notice that all the APs have the same length ($length(BGpath)$), 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 = set of all first partial APs = $\{AP_{1..1}\}$
for $i = 2$ **to** $length(BGpath)$ **do**:
  for all $AP_{1..i-1} \in L$ **do:** *// Expansion of each partial AP in L*
    $L = L - AP_{1..i-1}$
    $AP_{1..i} = AP_{1..i-1} \cup ComputeNextAction(AP_{1..i-1})$
    $L = L \cup AP_{1..i}$
  endfor
  $max\_value = max(h(AP_{1..i})), \forall AP_{1..i} \in L$
  $L = L - \{AP_{1..i} : AP_{1..i} \in L \wedge h(AP_{1..i}) < max\_value\}$
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, adding auxiliar actions if it is necessary. Those plans with a lower number of executable actions are rejected.

Selection of the action to be executed

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.

Then, these 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*[3], *Zeno-Travel*[4], $DriverLog$[4], etc. Flexible orders are very useful, for example, to order the *load* and *unload* actions in transportation domains before moving the involved vehicle.
- *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 like, for example, $BlocksWorld$[3], *Depots*[4], $FreeCell$[3], 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 reject those branches with a lower number of executable actions, etc.

The first action of the selected branch is inserted at the end of the final plan. The plan generation finishes when all the goals are achieved.

Results

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 [5]. 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 following graphics (Figures 2 and 3) show a comparative between SimPlanner planner and FF-Speed for the *Depots* and *Satellite* domains. An additional serie showing the time that SimPlanner takes to compute the first action

[3] Domains used in the IPC 2000 (http://www.cs.toronto.edu/aips2000)
[4] Domains used in the IPC 2002 (http://www.dur.ac.uk/d.p.long/competition.html)
[5] Full results of IPC2002 available at http://www.dur.ac.uk/d.p.long/competition.html

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 tenths 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.)

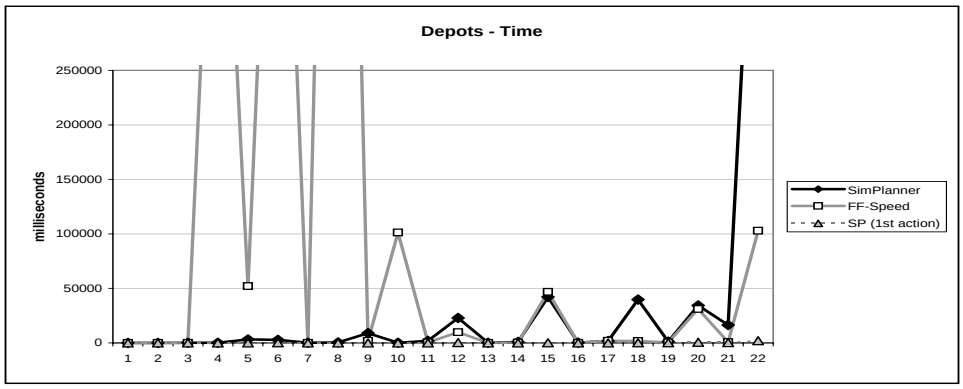

Fig. 2. Time for the *Depots* domain

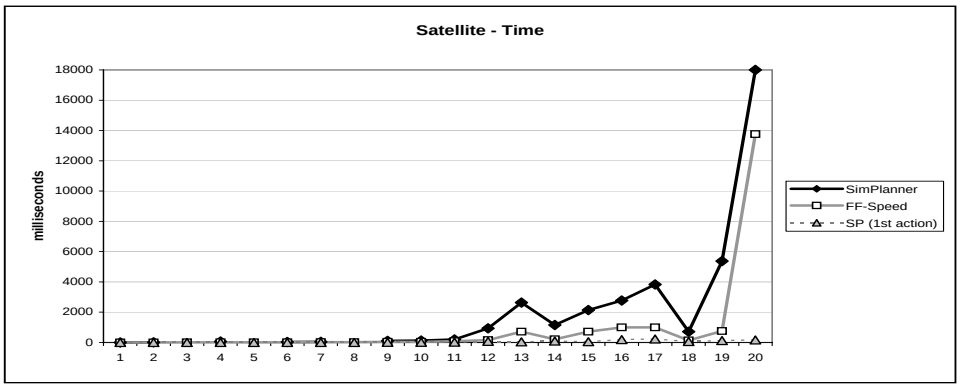

Fig. 3. Time for the *Satellite* domain

With regard to the quality of plans, in general SimPlanner produces longer plans than FF-Speed (Figure 4). 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 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.

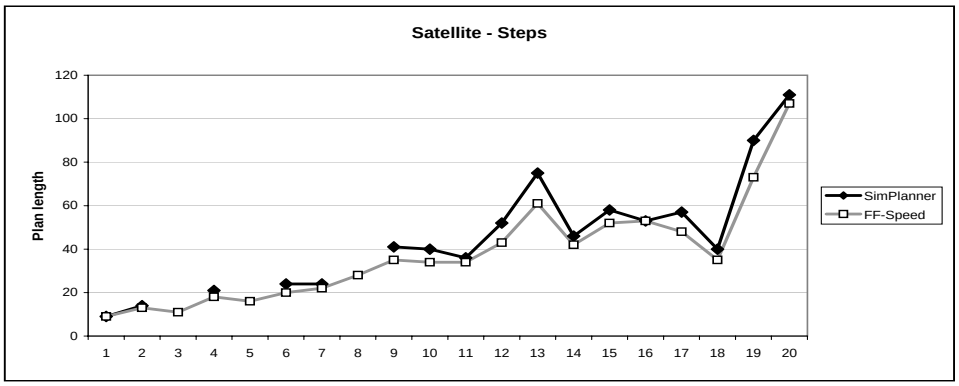

Fig. 4. Quality of the plans for the *Satellite* domain

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 (like batteries, fuel, etc.) are necessary. In these domains (like, for example, intelligent control of robots) 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.

References

1. J.A. Ambros-Ingerson and S. Steel, 'Integrating planning, execution and monitoring', *Proceedings of the 7th National Conference on Artificial Intelligence AAAI-88*, 83–88, (1988).

2. E.M. Atkins, E.H. Durfee, and K.G. Shin, 'Detecting and reacting to unplanned-for world states', *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, 1–7, (1996).
3. A. Blum and M. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**, 281–300, (1997).
4. J. Blythe, 'Planning with external events', *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, 94–101, (1994).
5. G. De Giacomo and R. Reiter, 'Execution monitoring of high-level robot programs', *Proceedings of Principles of Knowledge Representation and Reasoning*, 453–465, (1998).
6. T.L. Dean and M. Boddy, 'An analysis of time-dependent planning', *Proceedings of the 7th National Conference on Artificial Intelligence AAAI-88*, 49–54, (1988).
7. M. Drummond, K. Swanson, J. Bresina, and R. Levinson, 'Reaction-first search', *Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, 1408–1414, (1993).
8. M. Fox and L. Derek, 'Pddl2.1: An extension to pddl for expressing temporal planning domains', *available at http://www.dur.ac.uk/d.p.long/IPC/pddl.html*, (2002).
9. K.Z. Haigh and M. Veloso, 'Interleaving planning and robot execution for asynchronous user requests', *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, 35–44, (1996).
10. P. Haslum and H. Geffner, 'Admissible heuristics for optimal planning', *International Conference on AI Planning and Scheduling*, 140–149, (2000).
11. J. Hoffman and B. Nebel, 'The ff planning system: Fast planning generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).
12. Y. Lespérance and H.K. Ng, 'Integrating planning into reactive high-level robot programs', *Proceedings of the 2nd International Cognitive Robotics Workshop*, 49–54, (2000).
13. H. Levesque, 'What is planning in the presence of sensing?', *Proceedings AAAI*, 1139–1146, (1996).
14. T.S. Li and J.C. Latombe, 'On-line manipulation planning for two robot arms in a dynamic environment', *In Proceedings of the 12th Annual IEEE International Conference on Robotics and Automation*, 1048–1055, (1995).
15. I.R. Nourbakhsh, *Interleaving Planning and Execution for Autonomous Robots*, Kluwer Academy Publishers, 1997.
16. E. Onaindia, O. Sapena, L. Sebastia, and E. Marzal, 'Simplanner: an execution-monitoring system for replanning in dynamic worlds', *Proceedings of EPIA-01, Lecture Notes in Computer Science*, 393–400, (2001).
17. Z. Shiller, 'On-line sub-optimal obstacle avoidance', *International Journal of Robotics Research*, 480–497, (2000).
18. F. Terpstra, A. Visser, and B. Hertzberger, 'An on-line planner for marie', *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science (AICS2001)*, 199–209, (2001).
19. D.S. Weld, 'An introduction to least commitment planning', *AI Magazine*, **15**(4), (1994).