Total Order Planning is More Efficient than we Thought

Vincent VIDAL, Pierre RÉGNIER

Department of "Raisonnement et décision", IRIT, Paul Sabatier University
118 route de Narbonne
31062 Toulouse cedex, FRANCE
E-mail: {vvidal, regnier}@irit.fr

Abstract

In this paper, we present VVPLAN, a planner based on a classical state space search algorithm. The language used for domain and problem representation is ADL (Pednault 1989). We have compared VVPLAN to UCPOP (Penberthy and Weld 1992)(Weld 1994), a planner that admits the same representation language. Our experiments prove that such an algorithm is often more efficient than a planner based on a search in the space of partial plans. This result is achieved as soon as we introduce in VVPLAN's algorithm a loop test relating to previously visited states. In particular domains, VVPLAN can also outperform IPP (Koehler et al. 1997), which makes a planning graph analysis as GRAPHPLAN. We present here the details of our comparison with UCPOP, the results we obtain and our conclusions.

1 Introduction

1.1 Preliminaries

Planning has represented an important part of Artificial Intelligence for almost forty years and has received numerous developments. To solve more and more difficult problems, especially in robotics and productics domains, it has improved its formalism of representation and its algorithms. So, various planners have been developed. Among them, the most recent are commonly considered as more efficient than the eldest. The most employed techniques are search in the space of states, search in the space of partial plans and planning graph analysis. The first one — the eldest, (Fikes and Nilsson 1971) — is based on backward or forward state space search and the second one — more recent, (Chapman 1987) — on refinement strategies in space of partial plans (Kambhampati and Srivastava 1995), (Kambhampati 1997). Recently, the planners GRAPHPLAN (Blum and Furst 1997), then IPP (Koehler et al. 1997) have provided excellent results analyzing planning graphs.

We detail here an experimental study about VVPLAN, a forward state space planner for the ADL language. The numerous results we have achieved allow us to discuss the common opinion of the planning community "total-order planners are less efficient than partial-order planners". Indeed, the overwhelming majority of works that support that point of view does not take into account the knowledge about the current state of the world (which is realized planning forward). A total order planner provides good performances as soon as we use this knowledge to prune the search tree. These results complete other works as those of (Veloso and Blythe 1994), (Bacchus and Kabanza 1995) and HSP (www.ldc.usb.ve/~hector), (Bonet, Loerincs and Geffner 1997), demonstrating that state space search can be efficient in a lot of problems.

1.2 The current opinion

At present, the widespread belief in the planning community is that planning in the space of partial plans is more efficient than planning in the space of totally ordered plans or than planning in the space of states. So, in several important papers, we can quote numerous paragraphs where the authors give their opinion about total order and partial order planning according to this belief:

> (Penberthy and Weld 1992): Since consensus suggests that partial order planning is preferable to total order approaches (…), we pondered the void in the space of rigorous planners.
>
> (Kambhampati and Chen 1993): In contrast, the common wisdom in the planning community (…), has held that search in the space of plans, especially in the space of partially ordered plans provides a more flexible and efficient means of plan generation.
>
> (Kambhampati 1995): The conventional wisdom of the planning community, supported to a large extent by the recent analytical and empirical studies (…), holds that searching in the space of plans provides a more flexible and efficient framework for planning.
>
> (Blum and Furst 1995): It may seem puzzling that an extra level of commitment would lead to a fast planner, especially given the success enjoyed by least-commitment planners (…).
>
> (Bacchus and Kabanza 1995): The choice between the various search spaces has been the subject of much recent inquiry (…), with current consensus seemingly converging on the space of partially ordered plans (…).

These opinions are fundamentally based on several experimental studies that conclude to the superiority of

planning in the space of partially ordered plans on *planning in the space of totally ordered plans*: (Minton, Bresina and Drummond 1991), (Minton et al. 1992), (Minton, Bresina and Drummond 1994) and (Barrett and Weld 1994). Generally, those who quote these papers wrongly assimilate total order approaches and state space planning. In spite of this almost unanimous point of view (about the supposed efficiency of the partial order approach), two other papers — not well known and opposed to the precedent ones — really compare partial order planning and state space planning: (Veloso and Blythe 1994) demonstrate that some problems of particular domains can be solved more efficiently planning backward in the space of states than using partial order approaches. (Bacchus and Kabanza 1995) also compare, to its best advantage, the planner TLPLAN (forward state space planner, special language to express search control knowledge) to the planner SNLP (partial order planner).

Systematic comparisons between different planners begin to be dealt in controlled competitions — the first one has been organized during the last congress AIPS'98 (www.cs.yale.edu/HTML/YALE/CS/HyPlans/mcdermott.html). These experiments demonstrate the superiority of the GRAPHPLAN-like algorithms. This last generation of planners generally outperforms the conventional methods in most cases.

The experimental studies we detail here clearly demonstrate that the question is far from being solved, at least concerning the comparison between partial plans planning and planning in the space of states. First of all, we are going to sum up the papers we have quoted before: (Minton, Bresina and Drummond 1994) — revise and complete (Minton, Bresina and Drummond 1991), (Minton et al. 1992) — (Barrett and Weld 1994) which give a complementary and interesting viewpoint about the classification of planning domains, and (Veloso and Blythe 1994). Afterwards (cf. § 3), we will demonstrate that comparing UCPOP (Penberthy and Weld 1992), which is a reference for partial order planning, with VVPLAN, a forward state space planner, tests prove superior performances for the latter as soon as we prune the search tree thanks to the knowledge on already visited states. These results, which are in opposition to the common point of view, can be easily explained because most of the previous studies do not take into account this knowledge.

2 Main related works

(Minton, Bresina and Drummond 1994) compare a planner that searches in a space of partially ordered plans and a planner that searches in a space of totally ordered plans to determine the factors influencing performances and their respective importance (dimension of the search space, time cost of a refinement per plan, role of search strategies and heuristics, description language…). These algorithms are tested on a large set of randomly generated problems in the blocks world domain for various versions and heuristics. Performance criteria are the number of developed nodes and CPU time. These works demonstrate that in the blocks world domain and with or without domain independent heuristics, the partial-order planner UA is more efficient than the total-order planner TO.

(Barrett and Weld 1994) compare three planning algorithms which are tested on several artificial domains: POCL (space of partially ordered plans), TOCL (space of totally ordered plans, pre-ordering tractability refinements) and TOPI (space of totally ordered plans, adds operators to the head of the plan and is equivalent to a backward state space planner). They propose an extension of Korf's taxonomy of subgoal collections (Korf 1987) for planning in the space of partial plans[1]. Domains are built in order to present independent, trivially serializable or laboriously serializable subgoals for the different algorithms. Performances are estimated in CPU time, according to the number of subgoals. In all the domains, POCL and TOCL perform significantly better than TOPI, POCL being itself more efficient than TOCL. Those results support the conclusions of (Minton, Bresina and Drummond 1994). Afterwards, POCL and TOCL are tested on the fairly complex "Tyre world" domain which can not be solved in its initial version neither by POCL, nor by TOCL because of the laboriously serializable subgoals. If subgoals are ordered to make them serializable, the problem is then solved by POCL and TOCL (POCL being much more efficient). TOPI always fails.

(Veloso and Blythe 1994) compare SNLP (planning in a space of partially ordered plans) with PRODIGY (planning in a space of totally ordered plans, computation of the current state of the world to control the search). They define the linkability[2] property in order to predict the more efficient algorithm for solving a particular problem. SNLP is quasi identical to POCL, and PRODIGY is based on an algorithm similar to TOPI (backward planning in a space of states with possible computation of the current state of the world to control the search). These algorithms are tested on randomly generated problems from several artificial domains specially built to present, or not, the linkability property. Performances are estimated in CPU time, according to the number of subgoals. This method is similar to those of (Barrett and Weld 1994), except for the domain property. This time, results are favorable to total order planning in accordance with the selected criteria (CPU time and algorithms linkability property for the tested domains). In all cases, PRODIGY outperforms SNLP. This conclusion is *absolutely contradictory to the two previous studies*. The main reason is the use of a simulated world state which allows a backtrack reduction on the studied domains. In all the tested domains, goals are

[1] For example, in a given domain, problems may have trivially serializable subgoals for an algorithm and laboriously serializable for another one.

[2] This property is related to the order employed to post interval protection constraints in partial order planning: when this order is wrongly chosen, the algorithm must backtrack and works less efficiently.

trivially linkable for PRODIGY (solved in polynomial time according to the number of subgoals) and laboriously linkable for SNLP (solved in exponential time according to the number of subgoals). This property is therefore rather favorable to total order planning. These conclusions widely moderate those of the two previous papers.

See also (Kambhampati 1994) and (Kambhampati and Srivastava 1995), (Kambhampati and al. 1996) which detail a complete comparison of different plan space planning algorithms. All these different works show the following points:

- Nor the superiority of partial order on total order planning is proved neither the contrary. We can find domains that take advantage of both techniques.
- An ambiguity remains concerning state space planning: an algorithm that searches in a space of totally ordered plans like TOPI is theoretically equivalent to a backward state space search algorithm, but it does not use the knowledge on the current state of the world to control the search. When this is done, as with PRODIGY (Veloso and Blythe 1994) or TLPLAN (Bacchus and Kabanza 1995), results are widely modified.
- Nor the linkability neither the serialization are sufficient criteria to evaluate a planner. Some other criteria should be employed and then their respective influences should be studied.
- No comparison between partial order planning and forward state space planning has been done using already produced states to prune the search tree. The main interest of (Bacchus and Kabanza 1995) is the description of a special language employed to control the search, using domain dependent knowledge.

3 Comparison UCPOP / VVPLAN

3.1 Language and equipment

We have compared the UCPOP planner (version 4.1) with VVPLAN, a planner based on a state space search algorithm. Both planners used ADL to represent problems. We performed our tests in identical conditions: same programming language (CMU Common Lisp) and same equipment (PC with linux, CPU Intel Pentium 200 Mhz). Problems we have used are the 39 classical ones supplied with UCPOP. In order to reduce the influence of the system (particularly the Lisp garbage collecting influence), each has been tested several times: 100 times concerning problems solved in less than 0,1 seconds, 10 times concerning problems solved in 1 or 2 seconds, and only one, concerning the others. These tests clearly point out the greatest efficiency of VVPLAN as soon as we introduce a test based on memoization (called state loop control).

3.2 The compared algorithms

The search algorithm of UCPOP which is our reference remains the same during all the tests: an A* algorithm in the partial plans space using a domain independent heuristic. VVPLAN is built on a classical forward state space search algorithm. It accepts as input the current state of the world, the goal and the current plan. It begins with the initial state of the problem and the empty plan. The current plan is modified during the whole planning process. It is a totally ordered operator sequence. Three versions of this algorithm have been developed.

```
Refine-state-space (S: state, g: goal, P: plan)
1. Termination check:
   - if the goal g belongs to the state S, return
     P (P is a solution)
2. Action selection:
   - choose an action a using an operator in the
     set of operators, so that its preconditions
     belong to the state S
   - if no action can be applied to S, return fail
3. Action application:
   - apply the action a to S to reach a state S'
4. Recursive invocation:
   - Refine-state-space (S', g, P + ⟨ a ⟩[3])
```

Figure 1: The VVPLAN algorithm.

- **VVPLAN-breadth-first without state loop control:** is a classical breadth-first search algorithm; during steps 2 and 3 of the algorithm (cf. figure 1), all the possible states are created, applying all the possible actions to the current state of the world. Afterwards, those states are placed in a FIFO list. Search is carried on using the first state of the list as current state.
- **VVPLAN-breadth-first with state loop control:** this algorithm is similar to the previous one except for the further test: when the current state has been yet visited or is already situated in the FIFO list, it is not added to the latter. Even though this test seems to be expensive (numerous states must be memorized and the more numerous the created states are, the more long the test is), our implementation (numerical encoding of the states, hash-coding tables) produces good performances.
- **VVPLAN-A*:** is a classical A* heuristic search algorithm. The difference with the previous one relies on the order used to pick up the states from the list. To order the states, we do not use a FIFO method but the minimum value of a function f = g + h which adds: the cost g (length in number of steps) of the already realized path from the initial state to the current one, and the estimated value, using a heuristic[4] function h, of the length of the remaining path (in number of steps) from the current state to the final one.

This algorithm returns the shortest plan if and only if the heuristic function h minimizes the real cost of the remaining path. When several states have the same value for f, we use the optimistic interpretation: we pick up the

[3] The + operation joins two operators sequences: if $P = \langle a_1, \ldots, a_n \rangle$, then we have: $P + \langle a_{n+1} \rangle = \langle a_1, \ldots, a_n, a_{n+1} \rangle$.

[4] We use a domain independent heuristic h: the maximization of the number of solved subgoals in each state. This heuristic is generally admissible because in most of the tested domains, an action never adds at once more than one subgoal.

state with the greatest value for g i.e. both the most distant state from the initial one (g maximal) and the probably closest state to the goal (h minimal).

3.3 Description of the tests

They concern 39 different classical problems from 15 various domains, which are issued from numerous planners (STRIPS, PRODIGY, SNLP…). They have been adapted and improved thanks to the ADL language to be used by UCPOP. Those problems are performed using the three successive versions of VVPLAN. Results are estimated in CPU time and in the number of created and developed nodes. Search fails when it exceeds 100,000 created nodes.

3.4 Results

Comparison UCPOP / VVPLAN-breadth-first without state loop control. As we can see in figure 2, this first comparison gives the advantage to UCPOP. It solves 32 problems among the 39 ones (82 %) compared with only 26 for VVPLAN (66 %). Concerning these 26 problems (solved by the two planners), 12 (46 %) are performed faster by UCPOP; among the 32 problems solved by UCPOP, 18 (56 %) are solved faster by UCPOP. The difference is not so important but VVPLAN fails on 13 problems compared to 7 for UCPOP. Consequently, when VVPLAN uses the breadth-first algorithm without state loop control, these results point out UCPOP's superiority.
These results seem to confirm those of (Minton, Bresina, Drummond 1994), but 14 problems are solved faster by VVPLAN. This points out that the domains used in these studies are not different enough and are most often in favor of partial order planning.

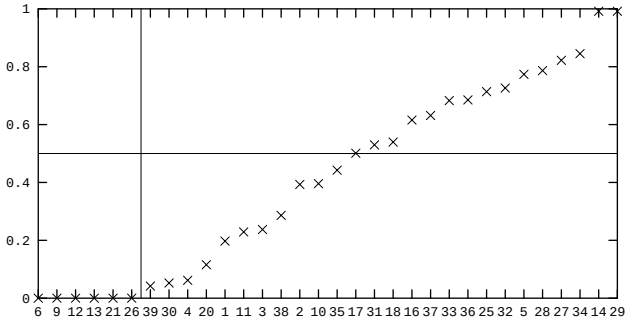

Figure 2: UCPOP / VVPLAN-breadth-first without state loop control (CPU time). *X-axis shows the tested problems (see figure 3 for the numbering of problems), and Y-axis the relative value between the CPU time of the two planners:* t_{UCPOP} / (t_{VVPLAN} + t_{UCPOP}). *Problems appear on the X-axis in the increasing order of the values, so as to obtain a usable curve. This order can vary from one curve to the other. The horizontal line (at 0.5) separates the zone in which UCPOP is more efficient (points bellow the line) from the zone in which VVPLAN is more efficient (points over the line). Problems at the left of the first vertical line (if there are some) remain unsolved by VVPLAN, and the ones at the right of the second line (if there are some) are not solved by UCPOP.*

1	fix1	11	Get-paid2	21	Monkey-test2	31	sched-test2a
2	fix2	12	get-paid3	22	monkey-test3	32	sussman-ano.
3	fix3	13	get-paid4	23	move-boxes	33	test-ferry
4	fix4	14	hanoi-3	24	prodigy-p22	34	tower-invert3
5	fix5	15	hanoi-4	25	prodigy-suss.	35	tower-invert4
6	fixa	16	mcd-suss.-an.	26	rat-insulin	36	uget-paid
7	fixb	17	mcd-sussman	27	road-test	37	uget-paid2
8	fixit	18	mcd-tower	28	r-test1	38	uget-paid3
9	fixit2	19	mcd-tower-in.	29	r-test2	39	uget-paid4
10	get-paid	20	monkey-test1	30	sched-test1a		

Figure 3: Numbering of problems.

Comparison UCPOP / VVPLAN-breadth-first with state loop control. When we add state loop control in VVPLAN's breadth-first search algorithm, results become totally different (see figure 4):

- All the 39 problems are now solved by VVPLAN, always compared with 32 (82%) problems for UCPOP. 23 problems among the 32 problems solved by the two planners are solved faster by VVPLAN (72 %, 9 more than with the first version of VVPLAN). On the average, VVPLAN solves those 32 problems 7 times faster than UCPOP (UCPOP 3.57 sec. against VVPLAN 0.48 sec.).
- VVPLAN creates far less nodes than UCPOP. Figure 5 (ratio of the created nodes: n_{UCPOP} / (n_{UCPOP} + n_{VVPLAN})) is similar to the CPU time graph. For 22 problems among the 32 ones solved by the two planners (69 %), UCPOP creates more nodes than VVPLAN.
- UCPOP also develops more nodes than VVPLAN in 30 of the 32 problems (93 %).

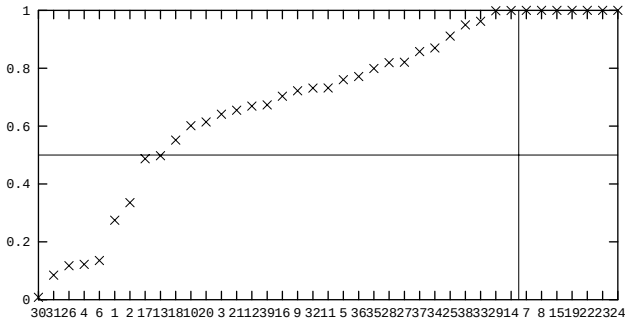

Figure 4: UCPOP vs. VVPLAN-breadth-first with state loop control (CPU time).

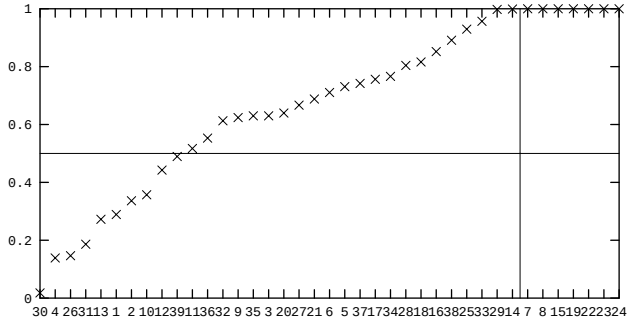

Figure 5: UCPOP vs. VVPLAN-breadth-first with state loop control (created nodes).

When these results are compared with the ones made without state loop control, we can point out that:

- On one hand, state space planning often develops a larger search space than partial order planning, which denies the current opinion.
- On the other hand, the average branching factor (number of created nodes / number of developed nodes) is higher for VVPLAN (6.38 against 1.44 for UCPOP), which confirms the established opinion.
- The difference of performance between these two versions of VVPLAN — without and with state loop control — demonstrates the interest of this test. The influence of the branching factor on the size of the search graph is minimized due to the tremendous number of pruned nodes.

Researchers using the results of (Barrett and Weld 1994) and (Minton, Bresina and Drummond 1994) to affirm that partial order planning is more efficient than state space planning make two mistakes:

- A total order planning algorithm as TOPI is not really equivalent to a state space planning algorithm. TOPI is indeed a very simple algorithm, without any improvement like state loop control. This kind of test can only be made by an algorithm which really computes states like VVPLAN. Even though this control seems to be expensive (in time and memory), it leads to really good performances thanks to the pruning of the search tree.
- (Barrett and Weld 1994) always point out that they have never obtained good performances for total order planning but only on their domains. Though they used a lot of different problems, it is not sufficient to prove that TOPI is always less efficient.

If we study the nature of the problems, we can notice:

- UCPOP's unsolved problems have a long plan-solution: on the average 12.6 actions compared with 5.2 actions for solved problems. Unsolved problems are complex ones and include a lot of operators and facts (problems "fixit", "fixb", "move-boxes"), or are problems with laboriously serializable subgoals (problem "hanoi4": the only possible serialization of subgoals is from the biggest disc to the smallest).
- The problem "fixit" is the famous one of the "Tyre World" domain (see § 2). According to (Barrett and Weld 1994), this problem can only be solved by a partial order planner like POCL or by a total order planner like TOCL (if subgoals are proposed in a serializable order) but never by a total order planner (assimilated to a state space planner) as TOPI. (Barrett and Weld 1994, p. 99) notice: "We took as our challenge, the problem of rendering this problem tractable". As it is demonstrated by our results, UCPOP does not solve this problem in its original form (without serialized subgoals) even though VVPLAN solves it in almost 6 sec, whatever is the subgoals order. VVPLAN creates 7,153 nodes to solve it, and UCPOP fails with more than 100,000 nodes.
- UCPOP's success on problems "sched-test1a", "sched-test2a" and "rat-insulin" can be explained because these problems have a lot of operators with few preconditions. This characteristic leads to an important branching factor for VVPLAN, but to a very little one for UCPOP (there are few constraints among actions). Partial plans developed by UCPOP have a lot of parallel actions. Thus, UCPOP's search space is far less important than VVPLAN's. It is typically the kind of problems that are solved faster by a partial order planner.
- Among the problems solved more efficiently by UCPOP, 4 are sub-problems of more complex ones that are not solved by UCPOP: "fix1", "fix2" and "fix4" are sub-problems of "fixit" and "fixa" is a sub-problem of "fixb". Sub-problem of a more complex one means a problem with an initial state to be reached in order to solve the complex problem, and with a goal to reach before solving the one of the complex problem. Despite the fact that UCPOP is efficient on easy problems of certain domains, it works laboriously on more complex problems in the same domains.

Comparison UCPOP / VVPLAN-A*. VVPLAN is now improved using an A* algorithm (with state loop control) and a classical domain-independent heuristic function: the maximization of the number of solved subgoals (in each state). The results we achieve are slightly better than the previous ones, even though it does not clearly appear on the figure 6 (CPU time). VVPLAN solves all the 39 problems, and UCPOP still solves 32 problems (82 %). 24 problems (one more than previously) are solved faster by VVPLAN (75 %) and the average CPU time (concerning these 32 problems) is now 0.46 sec. (against 0.48 sec. before). Furthermore, VVPLAN creates and develops slightly less nodes than in the precedent tests.

This curve does not appear to be very significant because it does not point out the improvement realized using the A* algorithm. This improvement appears if we take into account the 39 problems (cf. figure 7).

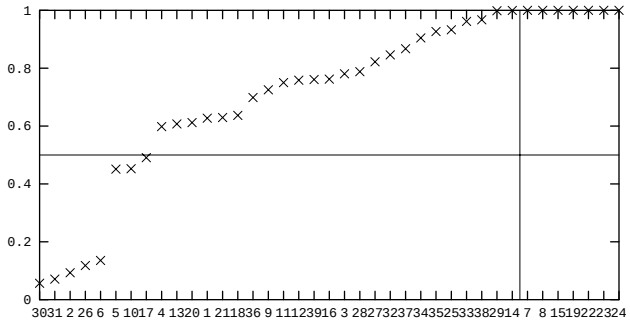

Figure 6: UCPOP / VVPLAN-A* with state loop control (CPU time).

We notice that:

- 31 problems (79 %) are solved faster by VVPLAN.

- The average CPU time (which was 2.10 sec. with VVPLAN-breadth-first) is now 1 sec. So, the time for evaluating the heuristic function does not penalize the overall performances. A* algorithm gives significantly better results on the 7 most difficult problems (which remain unsolved by UCPOP). Particularly, the CPU time needed by the two most difficult problems is now divided by 7 (problem "move-boxes") and by 6 (problem "prodigy-p22"). UCPOP's total average CPU time (109 sec.) is not really significant because it fails on 7 problems. However, this average CPU time is more than100 times higher than VVPLAN's.
- VVPLAN-A* creates (and develops) almost 30 % less nodes than VVPLAN-breadth-first when considering the 39 problems. This difference was nearly zero when considering the only 32 problems solved by the two planners. UCPOP creates 28 times more nodes and develops 98 times more nodes than VVPLAN-A*. This clearly points out VVPLAN-A*'s superiority on the performed examples.

		UCPOP	VVPLAN	
			state loop control	A*
CPU Time (sec.)	32 pbs.	3.57	0.48	0.46
	39 pbs.	> 109	2.10	1.00
Created nodes	32 pbs.	2,286	511	494
	39 pbs.	> 23,294	1,192	845
Developed nodes	32 pbs.	1,577	80	76
	39 pbs.	> 15,337	239	156

Figure 7: Results for all the problems.

5 Conclusion

In this paper, we have detailed an experimental comparison between VVPLAN, a forward state space planner, and the planner UCPOP. The results of these tests, allowed by the introduction of state loop control, show that VVPLAN outperforms UCPOP in most of the problems.

We also compared VVPLAN to IPP (results are not given here because of the lack of space). These tests demonstrate that some classes of problems can be solved more efficiently with a state space planner, giving an optimal solution. The efficiency of GRAPHPLAN-like planners is due to the internal qualities of this kind of algorithm, but this result is achieved against the quality of the solution. The latter is no more optimal in the number of actions, but in the number of time steps (a time step can hold several actions). We can suppose that problems of some domains, with no parallelism, a restricted branching factor and numerous redundant states, can be solved faster using a state space planner.

Finally, our feeling is that no planner outperforms all the others, in every domain. The differences of performance we observed seem to come essentially from some characteristics of the domains. It could be interesting to systematically try to characterize the numerous properties of domains (serialization of subgoals, linkability, number of operators, number of preconditions, effects and interactions…) to associate to each domain the probably most efficient algorithm.

Acknowledgements

This research was greatly improved by discussions and comments by E. Jacopin and S. Souville. We also thank the anonymous reviewers of this paper.

References

Bacchus F.; Kabanza F.; 1995. "Using temporal logic to control search in a forward chaining planner". EWSP'95.

Barrett A.; Weld D.S.; 1994. "Partial-order planning: evaluating possible efficiency gains". Artificial Intelligence, 71-112.

Blum A.; Furst M.; 1995. "Fast planning through planning graph analysis". IJCAI'95.

Blum A.; Furst M.; 1997. "Fast planning through planning graph analysis". Artificial Intelligence, 281-300.

Bonet B.; Loerincs G.; Geffner H.; 1997. "A robust and fast action selection mechanism for planning". AAAI'97.

Chapman D.; 1987. "Planning for conjunctive goals". Artificial Intelligence, 333-377.

Fikes R.; Nilsson N.; 1971. "STRIPS: A new approach to the application of theorem proving to problem solving" . Artificial Intelligence, 189-208.

Kambhampati S.; 1995. "Admissible pruning strategies based on plan minimality for plan-space planning". IJCAI'95.

Kambhampati S.; Srivastava B.; 1995. "Universal Classical Planner: An algorithm for unifying State-space and Plan-space planning". ECP'95.

Kambhampati S. and al.; 1996. "A Candidate Set based analysis of subgoals interactions in conjunctive goal planning". AIPS'96.

Kambhampati S.; 1997. "Refinement planning as a unifying framework for plan synthesis". AI Magazine, 67-97.

Kambhampati S.; Chen J.; 1993. "Relative utility of EBG based plan reuse in partial ordering vs. total ordering planning". AAAI'93.

Koehler J.; Nebel B.; Hoffmann J.; Dimopoulos Y.; 1997. "Extending Planning Graphs to an ADL Subset". ECP'97.

Korf R.E.; 1987. "Planning as search: a quantitative approach". Artificial intelligence, 65-88.

Minton S.; Bresina J.; Drummond M.; 1991. "Commitment strategies in planning: a comparative analysis". IJCAI'91.

Minton S.; Bresina J.; Drummond M.; Phillips A.; 1992. "Total order vs. partial order planning: factors influencing performance". KR'92.

Minton S.; Bresina J.; Drummond M.; 1994. "Total order and partial order planning: a comparative analysis". Artificial Intelligence, 71-111.

Pednault E.; 1989. "ADL: exploring the middle ground between STRIPS and the situation calculus". KR'89.

Penberthy J.S.; Weld D.S.; 1992. "UCPOP: a sound, complete, partial order planner for ADL". KR'92.

Veloso M.; Blythe J.; 1994. "Linkability: examining causal link commitments in partial-order planning". AIPS'94.

Weld D.S.; 1994. "An introduction to least commitment planning". AI Magazine, 27-61.