Searching Optimal Parallel Plans: A Filtering and Decomposition Approach

Guillaume Gabriel, Stéphane Grandcolas

Searching Optimal Parallel Plans: A Filtering and Decomposition Approach

Guillaume Gabriel and Stéphane Grandcolas
LSIS – UMR CNRS 6168
Avenue Escadrille Normandie-Niemen
F-13397 Marseille Cedex 20 - France
guillaume.gabriel@lsis.org, stephane.grandcolas@lsis.org

Abstract

In the domain of planning, searching for optimal plans gives rise to many works. Most of the existing planners search for optimal parallel plans, that is plans which are optimal in the number of steps. FDP is an exception: it searches for optimal sequential plans, that is plans which are optimal in the number of actions. FDP implements a depth-first iterative-deepening search, using a CSP-like structure, decomposition rules, and filtering techniques similar to consistency-enforcing techniques.

We propose a new planner based on FDP, which generates optimal parallel plans. This planner has been compared on a set of selected problems with the well-known optimal parallel planner SATPLAN*, and with the planner FDP, so as to analyse the effects of the optimization criteria in the performances.*

1. Introduction

Searching optimal plans solving SAT formulas or constraint satisfaction problems (CSP) is a commonly used approach. Some planners use external solvers, and can take advantage of solvers improvments, see SATPLAN [7] or GP-CSP [4] for example, or enhance generic solvers with specific strategies like MAXPLAN [10]. Others use their own engine like DPPLAN [1] or J. Rintanen[11], thus controlling completely the search. FDP [5] belongs to this second category: it combines constraint satisfaction and planning specific techniques. Simple rules propagate removals during the search, as consistency enforcing procedures do, while heuristics help to detect dead-ends. FDP distinguishes from other planners since it searches sequential optimal plans. Intuitively this optimization criteria should make the search less efficient, since the plans are longer. In fact FDP uses an ordering constraint which aims to avoid redundant searches, discarding equivalent action sequences. Is this constraint efficient and can it bridge the gap are open questions.

FDP iterative-deepening strategy and removals propagation procedure can easily be adapted to optimal parallel planning. We have implemented this approach. The aim of this work was, first to compare FDP approach with parallel planners with the same optimization criteria (at IPC-5 FDP competes against parallel planners), and secondly to analyse the effects of the optimization criteria on the performances.

In this paper we address STRIPS[1]-like propositionnal planning problems. A problem consists, given a set of actions A, to find a sequence of *compatible actions* sets called a *plan*, which achieves the goals of a given set G when executed in a given initial state I. The actions are defined by their preconditions and effects ; the preconditions are signed fluents, and the effects are additions and deletions of fluents. Two actions are compatible if none of them deletes a precondition or an addition of the other, or adds a fluent which is a negative precondition of the other. The transition states are constructed starting from the initial state, adding the actions additions and removing their deletions. Fluents which are not int the effects of the actions remain unchanged. F denotes the set of the fluents which occur in I, G and in the definitions of the actions. A plan is *optimal* if there is no shorter plan.

In the first part of this paper we present the planner FDP. Then we define the planning-structures that represent the planning problems and we introduce the consistency rules. Finally we describe a search procedure for optimal parallel plans, and we report some experimental results.

[1] Stanford Research Institute Problem Solver

2. FDP

Depth-first iterative-deepening [8] is a well-known strategy for finding optimal solutions in a state space. The algorithm consists in performing successive searches at limited depths, while no solution is encountered. The bound of the depth is incremented after each unsuccessfull search. Starting with a null initial bound guarantees that the first solution found is optimal. This approach is well suited for planning: states transitions correspond to the actions executions.

FDP implements a depth-first iterative-deepening strategy to search for optimal sequential plans. When searching for a plan of a given length k, FDP uses a CSP-like structure, a specific search procedure, and techniques closed to consistency enforcing procedures to propagate removals[9]. In FDP, the variables are assigned in a predefined order, starting from the initial state and progressing step by step towards the final state. This corresponds to the usual choice when assigning variables in a CSP, since the most constrained variables are the closest to the initial state. FDP then performs a forward chaining search: at each step one action is selected. After each assignment, inconsistent values are removed, and an heuristic helps to determine if the goals are reachable or not. To avoid redundant searches, independent actions which occurs consecutively have to respect a given ordering. Finally unsuccessfull searches are memorized. For further details we let the reader see [5].

3. Planning structures

Searching for a parallel plan of length k consists in *fixing* or *removing* actions in a *planning-structure* of length k (contrarily to FDP where only one action is selected at each step). Our planning-structures, like FDP planning-structures, are closed to Graphplan planning graph [2]. A planning-structure $\mathcal{S}$ is a set of variables organized into levels, that represent the fluents and the actions at each step of the plan. A planning-structure of size k comprises $k+1$ levels of facts variables, and k levels of actions variables. At each step i and for each fluent $f \in F$, a variable indicates if f is true, false or undefined at step i. At each step i and for each action $a \in A$, a variable indicates if a is possible, fixed or removed at step i. Levels are numbered from 0 to k for facts variables and from 1 to k for actions variables. A_i denotes the set of the possible actions at step i. X_i denotes the set of the fixed actions at step i. T (resp. F) designates the value true (resp. false) of fact variables. For convenience we will say that the fact f is true (resp. false) at step i if the value of the corresponding variable is true (resp. false).

Definition 1 (valid plan) *A valid plan of length k is a planning-structure $\mathcal{S}$ such that:*

- *the facts which are true (resp. false) in I are true (resp. false) at step 0 in $\mathcal{S}$,*
- *the goals of G are true at step k in $\mathcal{S}$,*
- *at each step i, $i > 0$, the actions which are not removed are fixed, their preconditions are true at step $i-1$ and their additions (resp. deletions) are true (resp. false) at step i,*
- *at each step i, $i < k$, each fact f which is true (resp. false) and which is not a deletion (resp. an addition) of a fixed action at step $i+1$, is true (resp. false) at step $i+1$,*
- *at each step i, fixed actions are compatible each other.*

Since the initial state is fully instantiated and actions at a same step have no opposite effects, then fact variables are defined at each step of a valid plan. One will easily verify from the definition that a valid plan is a solution for the planning problem and that to any solution plan corresponds a unique valid plan.

During the search some values are removed and these removals may propagate through the structure. For example, the deletion of the precondition of an action at step i implies that this action will be useless at step $i+1$, and then may be removed with no effect on the result of the search. Such values are said to be *inconsistent*.

Definition 2 (inconsistent values) *Given a planning-structure of size k and a fact $f \in F$, the value T (resp. F) of the variable corresponding to f at step i is inconsistent if one of the following conditions holds:*

- *$i > 0$ and $\exists a \in X_i$ s.t. a deletes f (resp. adds f),*
- *$i \leq k$ and $\exists a \in X_{i+1}$ s.t. a requires f to be false (resp. to be true),*
- *$i > 0$, $X_i = \emptyset$ and $\forall a \in A_i$, a deletes f (resp. adds f),*
- *$i \leq k$, $X_{i+1} = \emptyset$ and $\forall a \in A_{i+1}$, a requires f to be false (resp. to be true),*
- *$i > 0$, f is false (resp. true) at step $i-1$, and no action in $X_i \cup A_i$ adds f (resp. deletes f),*
- *$i \leq k$, f is false (resp. true) at step $i+1$ and no action in $X_{i+1} \cup A_{i+1}$ deletes f (resp. adds f).*

The action a is inconsistent at step i, $i > 0$, if one of the following conditions holds:

- *a precondition of a is falsified at step $i-1$,*
- *$\exists f \in F$ s. t. a adds f and f is false at step i,*
- *$\exists f \in F$ s. t. a deletes f and f is true at step i,*
- *$\exists a' \in X_i$ s. t. a and a' are incompatible.*

A fact value or an action which is not inconsistent is consistent.

The fourth rule for inconsistent values (if all actions in Ai have a common precondition and Xi is empty then the opposite of this precondition may be removed) is specific compared to J. Rintanen [11] or DPPLAN [1] approaches.

Definition 3 (consistent planning-structure) *A planning-structure is consistent if the actions and the values of the fact variables are consistent.*

As in FDP, during the search inconsistent actions and fact values are discarded, changing the status of the variables. The function **Make-Consistent** propagates changes through the planning structure using the consistency rules, as consistency enforcing procedures do with CSP. The function stops either with a consistent planning-structure and returns true, or because a change is impossible (a fact variable should be true and false), and returns false. In this last case no valid plan may be extracted from the planning-structure. Note that changes may propagate forward or backward in the planning-structure.

4. Searching plans

The objective is to find optimal parallel plans, that is plans which are minimal in the number of steps. The main process starts searching in a planning-structure of length one, then in a planning-structure of length two, and so on, until a solution is found or a given bound is reached (the system did not detect that no plan exists at the moment). The function **Search-Optimal-Plan** returns the minimal length of a plan for the problem P, if one exists of length at most max. While no plan is discovered, a planning-structure S of length k is contructed whose fact variables are undefined and actions variables are possible, the facts variable at the first step are instanciated with the values of the facts in I, the goals are fixed, and the removals are propagated. If the planning-structure is consistent the function **Search-Plan** is called to search a valid plan of length k.

```
function Search-Optimal-Plan(P, max)
{P is a planning problem}
1   for k := 1 to max do
2       S := Construct-Planning-Structure(P, k),
3       if Make-Consistent (S) then
4           if Search-Plan(S, k, 0) then
5               return k,
6   return NONE,

function Search-Plan(S, k, i)
{S is a k-steps planning-structure}
{A_i denotes the set of possible actions at step i in S}
1   if i = k then
2       return TRUE,
3   else
4       return Choose-Actions (S, k, A_i, i),

function Choose-Actions(S, k, A, i)
{S is a k-steps planning-structure}
{A_i denotes the set of possible actions at step i in S}
1   if A ∩ A_i = ∅ then
2       return Search-Plan(S, k, i + 1);
3   choose an action a ∈ A ∩ A_i,
4   S' := Fix-Action(a, i, S),
5   if Make-Consistent (S') then
6       if Choose-Actions(S', k, A − {a}, i) then
7           return TRUE,
8   S'' := Remove-Action(a, i, S),
9   if Make-Consistent (S'') then
10      if Choose-Actions(S'', k, A − {a}, i) then
11          return TRUE,
12  return FALSE.
```

Search-Plan performs a depth-first search, progressing forward from the first step. At each step i the function **Choose-Actions** is called to choose the fixed actions from the set A_i. The choice consists in fixing (lines 4-7) or removing (lines 8-11) a possible action. When there are no more possible action, the algorithm proceeds to the next step. If the planning-structure becomes inconsistent the procedure backtracks to previous choices. If the planning-structure becomes a valid plan the search stops with success.

Fix-Action(a, i, S) fixes the action a at step i in S: changes the status of the undefined preconditions and effects of a and removes A_i actions which are not compatible with a. **Remove-Action**(a, i, S) removes the action a at step i in the planning-structure S.

Remark that when **Search-Plan** proceeds to the i-th level of the planning-structure, the actions of the previous steps are either fixed or removed. Since the

initial state is fully instantiated and the fixed actions at a same step are compatible, then the fact variables before the step i are all true or false, defining a *state*. Memorizing these states when the search is un successfull or evaluating their capacity to achieve the goals are well-known techniques to improve search efficiency.

Failures memorizing. As in FDP, failures are memorized so as to avoid further similar searches: each time **Search-Plan** returns failure, the state at the current step is memorized in a hash table together with the number of remaining steps. Next time this situation will be encountered, the procedure will backtrack immediately. Note that if the planner did not perform a forward search, the memorization of the invalid states should be much more costly, since these states should be only partially instanciated.

Detecting goals unreachability. FDP implements a procedure which computes, each time a new level is explored, a maximal bound on the number of *unachieved goals* which may be added at the remaining steps. If the bound is less than the number of unachieved goals then the goals are unreachable. We have adapted this evaluation for parallel plans. For a given size p, we enumerate at each remaining step i the sets of compatible actions of size at most p (there are $O(|A_i|^p)$ sets at step i). We consider the number of unachieved goals which may be added by such sets when adding the remaining A_i actions compatible with its actions, and then deduce a maximal bound of the number of unachieved goals additions at step i. The sum of the bounds provides a criteria for goals unreachability. This heuristic is admissible, but it is basic compared to classic heuristics, see [6] for example. It focuses on the number of goals, not considering the way they may be achieved. It works well with sequential plans in which many steps are necessary to add many goals.

Irrelevant actions. Actions at last step which add no goal are useless for optimal solutions, and then can be removed. This property propagates backward (see [5]). As in FDP, irrelevant actions are removed from the planning-structure before searching.

5. Experimental results

We have implemented the approach described in this paper. In Figure 1 are reported the cpu times of FDP, SATPLAN (with the Siege V4 SAT solver), DPPlan (version 2.1), and different versions of our planner, denoted SPP, depending of the use of goals unreachability detection: no detection, or detection with compatible actions sets of size 1, 2 or 3. Experiments were all performed on the same machine, a linux computer with a Pentium 3 GHz processor and 1GB RAM, with a 1500 seconds timeout. Timeouts and planners crashes are represented with a $--$. Memory exceedings are indicated with a *m.e.*. We have compared the planners on a large selection of problems of IPC competitions (ICP3, IPC4 and IPC5). We have listed here a selection of problems which are representative of the different situations which were encountered. For each problem the optimal length, the makespan, the number of facts and the number of instanciated actions are reported.

First, one may notice that in most cases, length optimization is harder than makespan optimization. Theoretically this is the case when the optimal sequential plan is much longer than the parallel optimal one (at the last step of the search, the size of the search space is $O(2^{n^k})$ for a parallel plan and $O(n^k)$ for a sequential plan, where k is the length of the planning structure and n the number of actions). Nervertheless there are some exceptions, like PSR and Openstacks problems and Storage-13. These are problems with few facts, few actions, and large makespans. Openstacks problems and Storage-13 are particular since there is only one action per step in their solutions. Parallel planners are then penalized: they make useless verifications of the compatibility between the actions, and FDP takes advantage of its specific consistency rule which forces, if the value of a fact f changes at step i, each possible action at step i to have this change in its effects.

To evaluate the parallel version of FDP, we choose SATPLAN as reference (it dominated among the optimal parallel planners at IPC-4 and IPC-5 planning competitions). On many problems SATPLAN outperforms SPP. This can be explained by the increasing efficiency of SAT solvers and their highly optimized implementations. In fact SPP results are good in the Freecell serie, problems with a lot of actions, and Openstacks problems, problems with few actions but long solutions. Cpu times are very closed on Storage and Trucks problems. In all the other series SATPLAN is much better, see PSR or Rover series for example. Compared to SPP, DPPlan is less efficient on all problems exept on a Rover problem, and the gap is sometimes very important. Since SPP and DPPlan approaches are very closed, this is probably due to the fact that DPPlan does not memorize failures during the search. We observe with SPP that this technique drastically improves the search in many cases.

Finally Figure 1 reports the cpu times detecting goals unreachability calculating the maximal numbers of goals which may be added at each remaining step. For very few problems (PSR series) the search has been improved. Indeed the generation of the action sets is very costly. Furthermore this heuristic causes very few

problem	actions	facts	span	length	FDP	SATPLAN	DPPLAN	SPP	1-sets	2-sets	3-sets	gain
Freecell3-4	1160	139	8	14	64.38	4.97	10.45	1.69	1.75	1.81	2.03	0.05
Freecell5-4	2086	225	13		–	–	*m.e.*	10.19	10.24	10.47	11.52	0.01
Freecell7-4	4943	318	15		–	*m.e.*	*m.e.*	354.49	354	358.57	382.26	0
Airport-14	347	493	26	60	59.23	2.96	28.71	3.17	3.2	3.18	3.18	0
Airport-16	498	630	27		–	5.86	–	42.17	42.18	42.14	42.08	0
Airport-19	488	840	30		–	13.06	–	391.37	385.92	387.65	385.39	0
PSR-S33	163	41	15	25	3.56	0.54	384.08	49.91	49.87	49.89	49.86	0
PSR-S37	112	56	25	33	39.44	17.46	–	162.68	157.3	125.68	122.39	27.36
PSR-S45	179	68	18	21	22.52	1.69	1222.78	1015.58	864.64	407.49	399.35	56.84
Openstacks-4	115	37	23	23	2.4	–	–	3.82	4.01	4.63	5.04	4.12
Openstacks-8	115	37	23	23	2.39	–	–	3.85	4.01	4.6	4.98	5.79
Pathways-4	153	120	8	17	2.37	0.12	6.43	0.03	0.04	0.04	0.04	0
Pathways-5	266	149	9		–	0.34	–	32.62	33.09	33.12	33.41	0
Rover-2435	148	113	7		–	0.07	0.04	3.39	3.52	4.47	9.22	0
Rover-2312	-		11		–	0.21	651.64	–	–	–	–	
Storage-9	390	115	7	11	0.97	0.55	24.48	0.43	0.45	0.64	1.85	3.1
Storage-11	460	146	11	17	21.16	95.58	–	14.95	15.41	25.83	141.96	0.46
Storage-12	690	164	9	16	136.53	37.52	–	64.36	66.25	162.48	1500	
Storage-13	282	181	18	18	18.32	202.9	–	69.36	72.32	94.11	185.35	1.52
Storage-14	564	203	11	19	355.07	35.32	568.58	114.47	119.45	197.26	854.25	5.69
TPP-5	38	71	7	19	1.36	0.02	0.01	0.02	0.03	0.03	0.04	0
TPP-6	156	115	9		–	0.06	0.07	137.01	138.37	145.93	173.63	0
Truck-2	336	117	14	17	0.8	2.66	385.89	0.83	0.85	0.87	0.91	0
Truck-3	789	191	16	20	25.75	71.59	–	33.1	36.48	124.08	1314.9	0
Truck-4	936	226	18	23	1019.93	570.4	–	656.7	700.02	–	–	
Truck-7	1104	275	18	23	485.4	724.28	–	411.97	422.85	477.32	682.02	0

Figure 1: CPU times on a series of selected problems (times in seconds)

backtracks. The last column of the table contains the percentages of gain of the total number of actions and facts removals (this number is similar to the number of constraints checks when solving CSP), when the 3-sets heuristic is used. The gains are very low, and in many cases the computation times for the heuristic are very important (see storage series for example). The heuristic produces important gains only in the PSR series, in which SPP is really unefficient. Mutual exclusions between actions should be considered to constrain the compatible actions sets and to reinforce the heuristic. This will be the taken into account in future work.

6. Conclusion and perspectives

At IPC-5 planning competition, FDP was the only sequential planner to compete. It takes the second place in three series. We have modified FDP search engine so as to generate optimal parallel plans. The experimental results that we report in this paper prove that searching for optimal parallel plans is much easier than searching for optimal sequential plans. We solved many instances which were not solved by FDP. Compared with SATPLAN, our depth-first iterative-deepening approach gives good results in some cases, in particular when SATPLAN generates huge formulas.

Further developments will concern the combination of parallel and sequential searches, so as to constrain the number of actions and the number of steps. This could serve to optimize parallel plans while limiting the number of actions, or to minimize the number of actions while limiting the number of steps, and could compare with J. Rintanen approach [3].

References

[1] M. Baioletti, S. Marcugini, and A. Milani. Dpplan: An algorithm for fast solutions extraction from a planning graph. In *AIPS*, pages 13–21, 2000.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. In *IJCAI*, pages 1636–1642, 1995.

[3] M. Büttner and J. Rintanen. Improving parallel planning with constraints on the number of operators. In *ICAPS*, pages 292–299, 2005.

[4] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence*, 132(2):151–182, 2001.

[5] S. Grandcolas and C. Pain-Barre. Filtering, decomposition and search space reduction for optimal sequential planning. In *AAAI*, pages 993–998, 2007.

[6] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

[7] H. A. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *IJCAI*, pages 318–325, 1999.

[8] R. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[9] A. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, pages 8:99–118, 1977.

[10] S. M. Majercik and M. L. Littman. Maxplan: A new approach to probabilistic planning. In *AAAI*, pages 86–93, 1998.

[11] J. Rintanen. A planning algorithm not based on directional search. In *KR*, pages 617–625, 1998.