The Knowledge Engineering Review, Vol. 00:0, 1–24. © 2004, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

Parallel heuristic search in forward partial-order planning

OSCAR SAPENA, ALEJANDRO TORREÑO and EVA ONAINDÍA

Department of Information Systems and Computation, Universitat Politècnica de València, València, Spain
E-mail: osapena,atorreno,onaindia@dsic.upv.es

Abstract

Most of the current top-performing planners are sequential planners that only handle total-order plans. Although this is a computationally efficient approach, the management of total-order plans restrict the choices of reasoning and thus the generation of flexible plans. In this paper we present FLAP2, a forward-chaining planner that follows the principles of the classical POCL (Partial-Order Causal-Link Planning) paradigm. Working with partial-order plans allows FLAP2 to easily manage the parallelism of the plans, which brings several advantages: more flexible executions, shorter plan durations (makespan) and an easy adaptation to support new features like temporal or multi-agent planning. However, one of the limitations of POCL planners is that they require far more computational effort to deal with the interactions that arise among actions. FLAP2 minimizes this overhead by applying several techniques that improve its performance: the combination of different state-based heuristics and the use of parallel processes to diversify the search in different directions when a plateau is found. To evaluate the performance of FLAP2, we have made a comparison with four state-of-the-art planners: SGPlan, YAHSP2, TFD and OPTIC. Experimental results show that FLAP2 presents a very acceptable trade-off between time and quality and a high coverage on the current planning benchmarks.

1 Introduction

Until the late 1990s, Partial-Order Planning (POP) was the most popular approach to AI planning. In this approach, based on the least-commitment philosophy, decisions about action orderings and parameter bindings are postponed until a decision must be taken. This is an attractive idea as avoiding premature commitments requires less backtracking during the search process. Nevertheless, the most recent total-order forward-chaining planners, such as LAMA (Richter and Westphal (2010)), Fast Downward Stone Soup-1 (Helmert et al. (2011)) or SGPlan (Chen et al. (2006)), have demonstrated to be more efficient than partial-order planners, mainly due to:

- Search states can be generated much faster as there is no need to check *threats* (conflicts) among actions.
- They can generate complete state information and take advantage of powerful state-based heuristics or domain-specific control.

However, the general move towards state space search ignores some important benefits of partial-order planning:

- A partial-order plan offers more flexibility in execution.
- The search can be easily guided to improve the action parallelism in the plan.
- It is a very suitable approach in multi-agent planning systems, either with loosely (Kvarnström (2011)) or tightly coupled (Torreño et al. (2012)) agents.

- It can easily be adapted to deal with temporal planning (Benton et al. (2012)).

These desirable properties have led many current researchers to adopt POP techniques and to dedicate their efforts to improve the performance of this planning approach.

In this paper we present FLAP2, a partial-order forward-chaining planner that follows the design principles of POP, except for the delayed parameter binding, thus keeping the benefits of this successful approach. In spite of the inevitable increase of the search cost, we will show that FLAP2 improves the performance of existing partial-order planners and that it is competitive against some total-order planners. Particularly, FLAP2 returns solutions that represent a good trade-off between time and quality and it also offers a high coverage on the current planning benchmarks.

In the remainder of the paper we present the related work, some background and the planning approach of FLAP2. Finally, we present an empirical evaluation of the performance of FLAP2 and we conclude with some final remarks.

2 Related work

Looking at the winners of the last International Planning Competitions (IPC'2011[1] and IPC'2008[2]), we can observe that the majority of planners participated in the sequential tracks. Fast Downward Stone Soup-1 Helmert et al. (2011), Selective Max (Domshlak et al. (2010)) and Merge and Shrink (Helmert et al. (2013)) are optimal sequential planners built upon the classical Fast Downward planning system (Helmert (2006)) based on heuristic search. LAMA (Richter and Westphal (2010)), FF(h^s_a) (Keyder and Geffner (2008)) and C^3 (Lipovetzky and Geffner (2009)) are also forward state-space search planners that use powerful heuristics and compute (often suboptimal) solution plans very rapidly.

Planners that generate partial-order plans are basically found in temporal planning like SGPlan (Chen et al. (2006)), Temporal Fast Downward (Eyerich et al. (2009)), DAE$_{YAHSP}$ (Khouadjia et al. (2013)), YAHSP2 (Vidal (2011)) and POPF2 (Coles et al. (2010)). Temporal planning requires the ability of dealing with action parallelism due to the existence of temporally overlapping durative actions. With the exception of POPF2, all of these planners are built upon the parading of sequential planning. SGPlan, for example, uses Metric-FF (Hoffmann (2002)) as a search engine, while DAE$_{YAHSP}$ and YAHSP2 are developed on top of the YAHSP planner (Vidal (2003)). These three planners need an additional module to parallelize the obtained sequential plans and to enforce the temporal constraints of the problem. This separation between action selection and scheduling is doomed to fail in temporally expressive domains and suffer from severe drawbacks in temporally simple problems, as choosing the wrong actions might render the final solutions to be purely sequential and therefore of very low quality.

The approach taken by Temporal Fast Downward (TFD) is to perform forward search in the space of time-stamped states, where at each search state either a new action can be started or time can be advanced to the end point of an already running action, thereby combining action selection and scheduling (Eyerich (2012)). This approach is usually very good in terms of quality but their coverage on current benchmarks is typically relatively low.

From the aforementioned planners, POPF2 is the only one that follows a partial-order planning approach. It is a forward planner that works with time, numbers and continuous effects. POPF2 records state information at each step of the plan (frontier state), like the negative interactions among the variable assignments, and updates the state accordingly. The frontier state is used to determine the set of applicable actions at each step of the plan. The late-commitment approach of POPF2 is based on delaying commitment to ordering decisions on the frontier state, thus ignoring other alternative choices that would come earlier, i.e. *before* the frontier state. Completeness, however, is ensured as search performs backtracking to find an alternative plan when necessary.

[1] http://www.plg.inf.uc3m.es/ipc2011-deterministic
[2] http://ipc.informatik.uni-freiburg.de/

OPTIC (Benton et al. (2012)) is the latest version of POPF2 and also handles soft constraints and preferences. The key of its good performance is the fast generation of the successor states during the search and the use of effective domain-independent heuristics. OPTIC yields high quality plans, although, computationally speaking, it is not that efficient as most of the sequential planners.

In this paper we present FLAP2, a partial-order forward-chaining planner that follows the design principles of POP. This approach is similar to the one of OPTIC, but introduces two important differences:

- OPTIC adds additional temporal constraints over the action to ensure that preconditions of the new actions are met in the frontier state. The approach of FLAP2 is more flexible as it does not commit to an action ordering if this is not required, just like traditional POCL planners do.
- FLAP2 can add new actions at any point in the current plan. OPTIC only adds actions after the frontier state, so that the new actions do not threaten the preconditions of earlier actions.

These two differences lead to a more flexible partial-order planner, although this improvement entails a higher computational effort to deal with the interactions among actions. However, FLAP2 outperforms OPTIC in many domains because it uses more sophisticated search methods and more powerful heuristics. Moreover, delaying commitment on the orderings of the actions allows FLAP2 to reach a solution from a higher number of search nodes, which also improves the search performance.

3 Background

For the purposes of this paper, we restrict ourselves to propositional planning tasks. A planning task is a tuple $T = \langle O, V, A, I, G \rangle$. O is a finite set of objects that model the elements of the planning domain over which the planning actions are applied. V is a finite set of state variables that model the states of the world. A state variable $v \in V$ is mapped to a finite domain of mutually exclusive values D_v. A value of a state variable in D_v corresponds to an object of the planning domain, that is, $\forall v \in V, D_v \subseteq O$. When a value is assigned to a state variable, the pair $\langle$variable, value$\rangle$ acts as a ground atom in propositional planning. A is the set of deterministic actions. I is the set of initial values assigned to the state variables and represents the initial state of the task. G is the set of goals of the task, i.e., the values the state variables are expected to take in the final state.

Definition 1 *(Fluent) A ground atom or fluent is a tuple of the form $\langle v, d \rangle$ where $v \in V$ and $d \in D_v$, which indicates that variable v takes the value d.*

Definition 2 *(Action) An action $a \in A$ is a tuple $\langle PRE(a), EFF(a) \rangle$ where $PRE(a) = \{p_1, \ldots, p_n\}$ is a set of fluents that represents the preconditions of a and $EFF(a) = \{e_1, \ldots, e_m\}$ is a set of fluents that represents the consequences of executing a.*

We define a partial-order plan for a planning task $T = \langle O, V, A, I, G \rangle$ as follows:

Definition 3 *(Partial-order plan) A partial-order plan is a tuple $\Pi = \langle \Delta, OR, CL \rangle$. $\Delta \subseteq A$ is the set of actions in Π. OR is a set of ordering constraints ($\prec$) on Δ. CL is a set of causal links over Δ. A causal link is of the form $a_i \xrightarrow{\langle v,d \rangle} a_j$, meaning that precondition $\langle v, d \rangle$ of $a_j \in \Delta$ is supported by an effect of $a_i \in \Delta$.*

This definition of a partial-order plan represents the mapping of a plan into a directed acyclic graph, where Δ represents the nodes of the graph (actions) and OR and CL are the sets of directed edges that describe the precedences and causal links among these actions, respectively.

The introduction of new actions in a partial plan may trigger the appearance of flaws. There are two types of flaws in a partial plan: preconditions that are not yet solved (or supported)

through a causal link and threats. A threat over a causal link $a_i \xrightarrow{\langle v,d \rangle} a_j$ is caused by an action a_k that is not ordered w.r.t. a_i or a_j and modifies the value of v, i.e. $\langle v, d' \rangle \in EFF(a_k) \wedge d' \neq d$, making the causal link unsafe. Threats are addressed by introducing either an ordering constraint $a_k \prec a_i$, which is called *demotion* because the causal link is posted after the threatening action, or an ordering $a_j \prec a_k$, which is called *promotion* as the causal link is placed before the threatening action.

We define a *flaw-free* plan as a threat-free partial plan in which the preconditions of all the actions are supported through causal links. Given a flaw-free partial-order plan Π, we compute the frontier state, S_Π, resulting from the execution of Π in the initial state I. More formally:

Definition 4 *(Frontier state) The frontier state S_Π of a flaw-free partial-order plan $\Pi = \langle \Delta, OR, CL \rangle$ is the set of fluents $\langle v, d \rangle$ achieved in Π by an action $a \in \Delta / \langle v, d \rangle \in EFF(a)$, such that any action $a' \in \Delta$ that modifies the value of v ($\langle v, d' \rangle \in EFF(a') / d \neq d'$) is not reachable from a by following the orderings and causal links in Π.*

The basic POP algorithm starts by building an initial minimal plan containing two fictitious actions: the initial action a_{init}, with no preconditions and $EFF(a_{init}) = I$, and the goal action a_{goal}, with no effects and $PRE(a_{goal}) = G$. The algorithm works by following the next three steps until a solution is found: 1) select the next subgoal to achieve, 2) choose an action to support the selected subgoal and 3) solve the *threats* that arise as a consequence of the variables value modification.

In the following section we describe the planning algorithm of FLAP2 as well as the necessary modifications to adapt a POP algorithm to support a forward search. In our effort to maintain all the benefits of this approach, we tried to keep the changes as minimal as possible.

4 Planning algorithm

FLAP2 is a modified version of FLAP planner (Sapena et al. (2013)). In the following subsections we briefly describe the planning approach of FLAP and the changes made in FLAP2 to improve its performance, respectively.

4.1 FLAP's working scheme

FLAP implements an A* search, as the standard textbook algorithm in (Russell and Norvig (2009)), guided by an evaluation function. A search node is a partial-order plan and the starting node is the initial initial plan $\Pi_0 = \langle \{a_{init}\}, \emptyset, \emptyset \rangle$. Although Π_0 does not contain the fictitious goal action a_{goal}, this action is available to be added to the plan as the rest of actions in A, i.e. $a_{goal} \in A$. In fact, a solution plan is found when a_{goal} is inserted in the plan.

FLAP follows two steps at each iteration of the search process until a solution plan is found: **a)** it selects the best node, Π_i, from the set of open nodes according to the evaluation function, and **b)** all possible successors of Π_i are generated, evaluated and added to the list of open nodes. FLAP considers that Π_j is a successor of a plan Π_i if the following conditions are met:

- Π_j adds a new action a_j to Π_i, i.e., $\Delta_j = \Delta_i \cup \{a_j\}$
- All preconditions of a_j are supported with actions in Π_i by inserting the corresponding causal links: $\exists a_i \xrightarrow{p} a_j \in CL_j, a_i \in \Delta_i, \forall p \in PRE(a_j)$.
- All threats in Π_j are solved through promotion or demotion by adding new ordering constraints; the result is that Π_j is a flaw-free plan.

The forward-search approach of FLAP allows to use state-based heuristics, which are much more informed than classical POP-based heuristics. In order to evaluate a partial-order plan Π, FLAP computes the frontier state S_Π. It uses three different heuristics:

- h_{DTG}. A Domain Transition Graph (DTG) of a state variable is a representation of the ways in which the variable can change its value (Helmert (2004)). Each transition is labeled with the necessary conditions for this to happen, i.e. the common preconditions to all the actions that induce the transition. These graphs are used to estimate the cost of the value transition required to support an action precondition, and the *Dijkstra* algorithm is applied to calculate the length of the shortest path in the DTG that causes the transition. The h_{DTG} heuristic returns the minimum number of actions in a relaxed plan, where delete effects are ignored, that achieves the problem goals from S_Π. Actions in the relaxed plans are selected according to the sum of the estimated cost of their preconditions.
- h_{FF}. FLAP also makes use of the traditional FF heuristic function h_{FF} (Hoffman and Nebel (2001)), which builds a relaxed plan by ignoring the delete effects of the actions and returns its number of actions. The actions of this plan are selected according to their levels in the relaxed planning graph.
- h_{LAND_DTG} and h_{LAND_FF}. Landmarks are fluents that must be achieved in every solution plan (Hoffmann et al. (2004); Sebastia et al. (2006)). FLAP computes a landmark graph and uses this information to calculate heuristic estimates: since all landmarks must be achieved in order to reach a goal, the goal distance can be estimated through the set of landmarks that still need to be achieved from the state being evaluated onwards. Once we have the set of non-supported landmarks, the heuristic value is the result of estimating the cost of reaching these landmarks with either h_{DTG} or h_{FF}. This way, FLAP has two versions of the landmarks heuristic, called h_{LAND_DTG} and h_{LAND_FF}, respectively.

For evaluating a plan $\Pi = \langle \Delta, OR, CL \rangle$, FLAP defines two different evaluation functions:

- $f_{FF}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND_FF}(\Pi) + w_3 * h_{FF}(S_\Pi)$
- $f_{DTG}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND_DTG}(\Pi) + w_3 * h_{DTG}(S_\Pi)$

$g(\Pi)$ measures the cost of Π in number of actions, i.e. $g(\Pi) = |\Delta|$. The weights in the two functions are set to $w_1 = 1$, $w_2 = 4$ and $w_3 = 2$. FLAP uses both evaluation functions to simultaneously explore different parts of the search space, thus defining two main search processes.

Additionally, a new A* search is started in parallel when one of the two main search processes is stuck in a plateau, i.e. the evaluation function does not improve after several iterations. The goal of this new search is not to escape from the plateau, but to find a solution plan starting from the frontier state of the best node found so far, as this node is more likely to be closer to a solution than the initial state. The parallel search is cancelled if the main search manages to leave the plateau.

FLAP planner is sound and complete since all possible successors are considered at each point and, when a_{goal} is added to the plan, the support of all problem goals as well as the plan consistency is guaranteed.

4.2 Performance improvements in FLAP2

In order to improve the performance of FLAP we performed an analysis of the search process, specifically of the behaviour of the heuristics in domains with different characteristics. This analysis is shown in the following subsection. Finally, in a second subsection, we describe the modifications introduced in FLAP2 according to the conclusions of the analysis.

4.2.1 Analysis of heuristics and the plateau escaping method.

Regarding h_{DTG}, we found that this heuristic is more informative than h_{FF} in planning domains where the state variables have rather large domains, containing multiple different values, and the DTGs of these variables are sparse graphs.

In Figure 1 we can observe an example of the DTGs of two variables: (*empty t1*) and (*at d1*). There are only two values, *true* and *false*, in the domain of (*empty t1*), meaning that the cabin

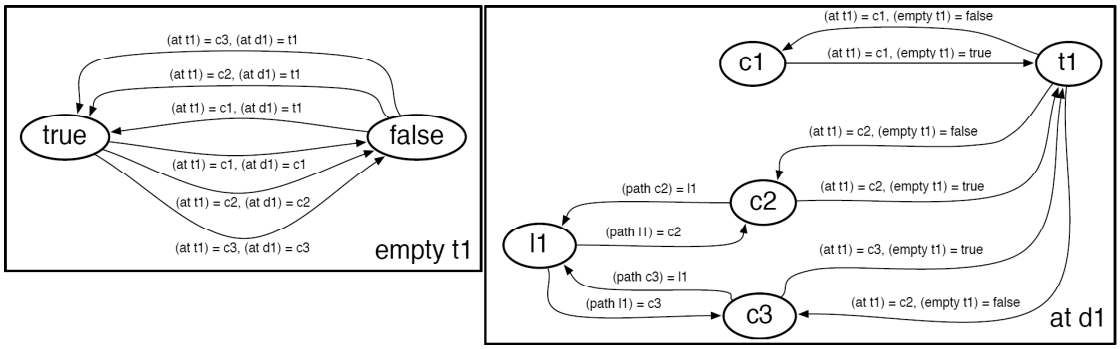


Figure 1 DTGs of variables (*empty t1*), the state of the cabin of truck *t1*, and (*at d1*), the location of driver *d1*, in a *DriverLog* problem example.

of the truck *t1* can be empty or not. On the contrary, the position of driver *d1* can take several different values: location 1 (*l1*), cities 1, 2 and 3 (*c1*, *c2* and *c3*) and truck 1 (*t1*). The values of h_{DTG} obtained from the DTG of variable (*empty t1*) are not very accurate because there is only one transition that makes the variable change from *true* to *false*, and this transition is derived by many different actions, particularly all actions in which *d1* boards *t1* at any possible city. Hence, selecting the action to be included in the relaxed plan to support this transition is not an easy task and a wrong decision would worsen the quality of the heuristic.

On the contrary, the DTG of variable (*at d1*) is more informative. For example, the path to change its value from *l1* to *c1* contains three transitions: $l1 \rightarrow c2 \rightarrow t1 \rightarrow c1$ or $l1 \rightarrow c3 \rightarrow t1 \rightarrow c1$, depending on the position of the truck. Moreover, each transition in the path is produced by a single action and thus the correct action is always selected by h_{DTG} when computing the relaxed graph. Our conclusion is that h_{DTG} performs slightly better than h_{FF} in transportation-like domains, such as *DriverLog* or *ZenoTravel*, where the DTGs of several variables are rather large sparse graphs. For the rest of domains, h_{FF} clearly outperforms h_{DTG}.

h_{DTG} also presents some limitations in non-reversible domains, where the effects of some actions cannot be undone. The search space of these domains may contain dead-ends, i.e., nodes with frontier states from which the problem goals are unreachable. h_{FF} is able to detect many of these dead-ends as it builds a relaxed planning graph at each node of the search tree: if any of the problem goals is not reachable in the relaxed graph, the node is a dead-end. On the contrary, h_{DTG} only detects a dead-end state if no transition path can be found in the DTGs that transforms the value of a variable into its final value. Then, h_{DTG} does not take into account the interactions between variables to detect dead-ends. This limitation can be alleviated by computing mutex fluents in a preprocessing stage, i.e. fluents that cannot be true in a state at the same time. Improvements in the h_{DTG} heuristic is an issue we want to address in future works.

On the other hand, the landmark-based heuristic, h_{LAND}, is very informative in domains which contain a large number of atomic landmarks. An atomic landmark, which is a single fluent that every solution plan must achieve at some point, is usually much more accurate than a disjunctive landmark since a disjunctive landmarks is less restrictive. In FLAP, h_{LAND} (both h_{LAND_FF} and h_{LAND_DTG}), is always used in combination with h_{FF} or h_{DTG}. However, we observed that, when the number of atomic landmarks is similar or greater than the number of disjunctive landmarks, h_{LAND} is informative enough to be used as a stand-alone heuristic.

These three heuristics (h_{DTG}, h_{FF} and h_{LAND}) assess the quality of a plan by estimating the number of actions required to reach the problem goals. However, this does not seem to be the most appropriate approach for a planner that works with concurrent actions. When dealing with partial-order plans, optimizing the plan duration (makespan) is always preferable if we aim to improve the plan parallelism. Even so, as we will see in the Experimental Results section, the quality of the plans generated by FLAP2 w.r.t. the makespan is quite good because it exploits

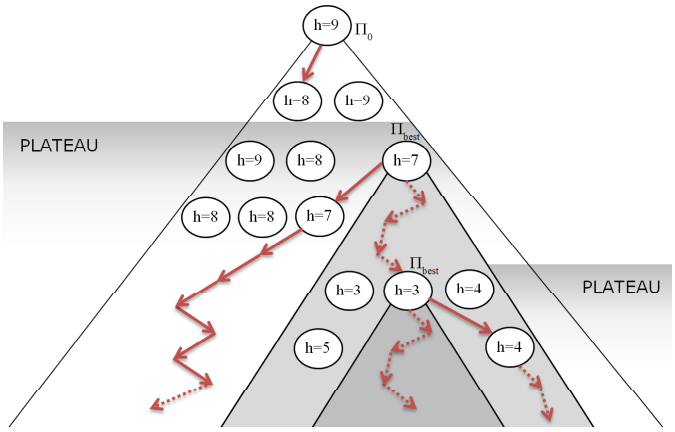


Figure 2 Parallel A* search processes for plateau escaping.

the advantages of working directly with concurrent actions. However, adapting the heuristics to evaluate the plans according to their makespan could significantly improve the quality of the solutions, a research line we intend to explore in the future.

Finally, we analyzed the plateau escaping mechanism of FLAP. The parallel search process started when one of the main search processes gets stuck in a plateau is not enough to solve some difficult problems as this new search may also get stuck in another plateau.

4.2.2 Modifications in the search process of FLAP.

Taking all the above considerations into account, we designed FLAP2 as follows. First of all, we check if sufficient information can be extracted from the landmarks graph. We define $\lambda = |disjunctive_landmarks|/|atomic_landmaks|$, i.e. the ratio between the number of disjunctive landmarks and the number of atomic landmarks; when no atomic landmarks are found, $\lambda = \infty$. We consider that there is enough information when $\lambda \leq 1.2$.

When h_{LAND} is not informative enough, $\lambda > 1.2$, FLAP2 starts a single main A* search with the f_{FF} evaluation function with $w_1 = 1$, $w_2 = 4$ and $w_3 = 2$. The weight for h_{LAND_FF}, w_2, is higher to make up for the poor heuristic values returned by h_{LAND}. Unlike FLAP, in FLAP2 we do not start a second main search with h_{DTG} because, as we said in the previous section, h_{DTG} is only worth using in transportation-like domains and thereby a general use of h_{DTG} does not compensate for the overhead in computation time and memory consumption. Consequently, h_{DTG} is only used in FLAP2 when search needs to be diversified due to the existence of a plateau.

The search process of FLAP2 uses a variable, Π_{best}, that stores the node with the best heuristic value found so far. Initially Π_{best} is set to the initial plan, i.e. $\Pi_{best} = \Pi_0$. When a search node with a better heuristic value than the one of Π_{best} is found, Π_{best} is updated to this node. We consider that the search is stuck in a plateau when Π_{best} has not been updated in several iterations. In this case, two new search processes are started from the frontier state of Π_{best} to increase the chances of escaping from the plateau. The first one uses f_{FF} and the second one the f_{DTG} evaluation function, both with the same weight values than the ones used for the main search. By using two new searches with different heuristic functions, we allow to diversify the search directions and find a plateau exit more effectively.

A *child* search works equally as the main search. In fact, when a child search finds a plateau, it also starts two new search processes. This behaviour can be observed in Figure 2. When a search manages to escape from a plateau, i.e. when a node with a heuristic value better than the value of Π_{best} is found, then its two child processes are terminated.

In the case that h_{LAND} is informative enough, $\lambda \leq 1.2$, FLAP2 starts a search process with f_{FF} and a second main A* search with the following evaluation function: $f_{LAND_FF}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND_FF}(\Pi)$, with $w_1 = 1$ and $w_2 = 1$. In this case, h_{LAND_FF} is used as a stand-alone

heuristic function. When a plateau is found, two child searches are started in the same way as for the case of $\lambda > 1.2$, but now we use f_{FF} with $w_1 = 1$, $w_2 = 1$ and $w_3 = 1$, and $f_{LAND_DTG}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND_DTG}(\Pi)$ with $w_1 = 1$ and $w_2 = 1$. This configuration has been fixed as the result of an extensive experimental analysis and it offers a good trade-off between search time and plan quality in most of the problems.

The mechanism of parallel searches implemented in FLAP2 yields very good results but it can lead to an exponential growth in the number of simultaneous processes. However, this problem does not usually occur in practice since the number of simultaneous search processes that exceeded the number of processing cores (8 in our test computer) only occurred in a few problems. Specifically, we tested FLAP2 in 244 problems from 10 different domains and only 7 of them required more than 8 search processes at the same time. And yet, this did not prevent FLAP2 from finding a solution plan for these problems.

5 Experimental results

In order to evaluate the performance of FLAP2, we selected four current top-performing planners that return parallel plans: SGPlan, YAHSP2, OPTIC and TFD. All of them are temporal planners as only this type of planners are currently able to synthesize plans with concurrent actions. Due to the different characteristics of these planners, we have divided this section in two subsections:

- Comparison of FLAP2 with SGPlan and YAHSP2, two sequential planners that apply a scheduler to parallelize the plans at a later stage. This approach is extremely fast but finds more difficulties in producing plans of good quality regarding the makespan.
- Comparison of FLAP2 with OPTIC and TFD, two planners that merge the action selection and the scheduling process. Working with partial-order planners allows to compute more flexible plans, with a better makespan, but slows down the search process.

In both cases, we selected six temporal domains from the International Planning Competitions (IPC), setting the duration of all actions to 1 as FLAP2 is still unable to work with durative actions. We observed that the behaviour of these planners varies greatly depending on the level of interaction between the problem goals. For this reason we selected three domains with strong dependencies between the goals, *BlocksWorld*, *Depots* and *DriverLog*, and three domains with rather independent goals, *Satellite*, *Rovers* and *ZenoTravel*.

Testing was performed on a 2.3 GHz i7 computer with 12 GB of memory running Ubuntu 64-bits. We only consider the first plan returned by the planners as most of them do not continue searching for better plans. Each experiment was limited to 30 minutes of wall-clock time.

5.1 FLAP2 vs. SGPlan and YAHSP2

Table 1 shows the number of solved problems and the average time employed by these planners to find the first solution. Average times are calculated considering only those problems that were solved by the three planners.

As it can be observed, FLAP2 solves more problems and shows a more stable behaviour. Both, SGPlan and YAHSP2 present some difficulties in domains with strong interactions between the

		FLAP2		SGPlan		YAHSP2	
Domain	Prob	Solved	Average time	Solved	Average time	Solved	Average time
BlocksWorld	34	34	0.40	22	5.80	34	57.78
Depots	20	20	1.99	19	0.15	16	121.24
DriverLog	20	20	3.38	17	1.02	20	0.11
Satellite	20	20	4.19	20	0.07	20	0.05
Rovers	20	20	4.21	20	0.04	20	0.04
ZenoTravel	20	20	6.91	20	0.23	20	0.16
Total	134	134	3.52	118	1.22	130	29.90

Table 1 Number of problems solved and average time (in seconds) of FLAP2, SGPlan and YAHSP2.

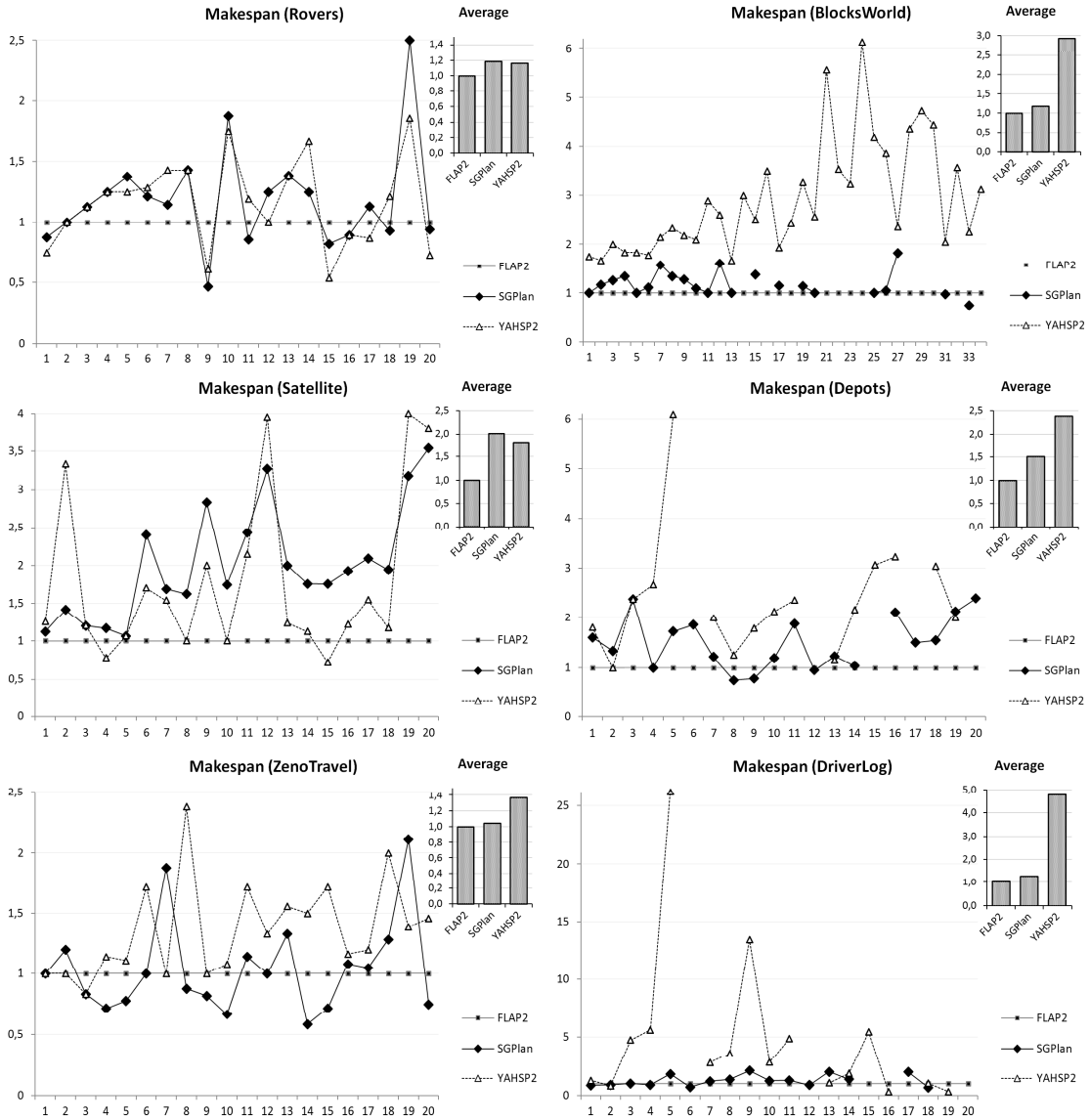


Figure 3 Makespan of the plans of SGPlan and YAHSP2, normalized by the makespan of the plans of FLAP2.

goals (*BlocksWorld*, *Depots* and *DriverLog*), but they are significantly faster in the other three domains. The landmarks heuristic and the plateau escaping mechanism of FLAP2 are very helpful to deal with strong dependencies among the goals. FLAP2 also easily solves the problems from the *Rovers*, *Satellite* and *ZenoTravel* domains, but the overhead to cope with threats among actions together with a higher branching factor prevents FLAP2 from being as faster as SGPlan or YASHP2 in these domains.

Regarding the plan quality, Figure 3 shows the makespan of the plans computed by the three planners. The results are normalized by the makespan of the plans obtained by FLAP2 for a better viewing. This way, a value of 2 indicates a plan with a makespan twice as much as the makespan of FLAP2, and a value of 0.5 a plan two times shorter.

In general, FLAP2 generates plans with better quality than SGPlan and YAHSP2. SGPlan produces slightly worse plans, 1.36 times longer in the six domains. The plan quality of YAHSP2 is much worse as the generated plans are 2.4 times longer than FLAP2 on average.

Domain	Prob	FLAP2 Solved	FLAP2 Average makespan	OPTIC Solved	OPTIC Average makespan	TFD Solved	TFD Average makespan
BlocksWorld	34	34	10.92	24	15.88	34	7.25
Depots	20	20	11.93	11	14.86	10	9.10
DriverLog	20	20	14.47	15	12.93	16	13.40
Satellite	20	20	17.00	16	11.50	20	14.25
Rovers	20	20	12.65	20	13.35	17	14.29
ZenoTravel	20	20	8.56	16	8.31	20	8.31
Total	134	134	12.59	102	12.81	117	11.10

Table 2 Number of problems solved and average makespan of FLAP2, OPTIC and TFD.

5.2 FLAP2 vs. OPTIC and TFD

Table 2 shows the number of solved problems and the average makespan of FLAP2, OPTIC and TFD. As it can be observed, FLAP2 also solves more problems than OPTIC and TFD. The average makespan is computed taking into account only those problems that were solved by the three planners. Regarding the makespan, FLAP2 is in a intermediate position between TFD, that produces plans of very good quality, and OPTIC.

In Figure 4 we show the computation time of FLAP2, OPTIC and TFD to find the first solution plan. For the average times shown in these figures, we considered only the problems that the three planners have managed to solve. FLAP2 is much faster than OPTIC in the *BlocksWorld*, *Depots*, *Satellite* and *ZenoTravel* domains. On the contrary, OPTIC is slightly faster than FLAP2 in the *Rovers* domain. On average, OPTIC is 113.94 times slower than FLAP2 in all the six domains. TFD is also slower than FLAP2, especially in the *Depots* and *DriverLog* domains. On average, TFD is 45.3 times slower than FLAP2 in all the six domains.

In summary, we can conclude that FLAP2 is very competitive in comparison with these four top-performing planners. It solves more problems than SGPlan, YAHSP2, OPTIC and TFD in the tested domains. FLAP2 also produces plans of better quality than the sequential planners SGPlan and YAHSP2, and is far more faster than OPTIC and TFD, planners that, like FLAP2, handle partial-order plans.

6 Conclusions

The flexibility of the Partial-Order Planning (POP) paradigm allows for the generation of high-quality parallel plans. However, current sequential planners outperform partial-order planners because they require less computational effort as they not need to cope with interactions among actions and can use very effective state-based heuristics.

In this paper we present FLAP2, an improved version a FLAP. FLAP is a forward partial-order planner that combines three different heuristics to guide the search and implements a novel plateau-escaping method that diversifies the search in different directions. FLAP2 changes the way the heuristics are combined and applies a recursive method to deal with plateaus, thus significantly improving the planning performance.

We compared FLAP2 with SGPlan, YAHSP2, OPTIC and Temporal Fast Downward (TFD), four top-performing planners that can generate plans with concurrent actions. Like FLAP2, OPTIC and TFD handle partial-order plans, combining the action selection and the scheduling processes. On the contrary, SGPlan and YAHSP2 are total-order planners that parallelize the computed plans at a later stage.

FLAP2 is the only one that was able to solve all the problems in the selected benchmark set. Regarding the makespan (plan duration), partial-order planners generate plans of much better quality than the total-order planners. Particularly, FLAP2 has shown to obtain plans of very good quality, only surpassed by TFD, which is able to produce plans with a slightly better makespan. As for the planning time, FLAP2 has shown to be competitive with the sequential planners, SGPlan and YAHSP2, especially in domains with strong interactions between the problem goals, and far more faster than the other partial-order planners, OPTIC and TFD.

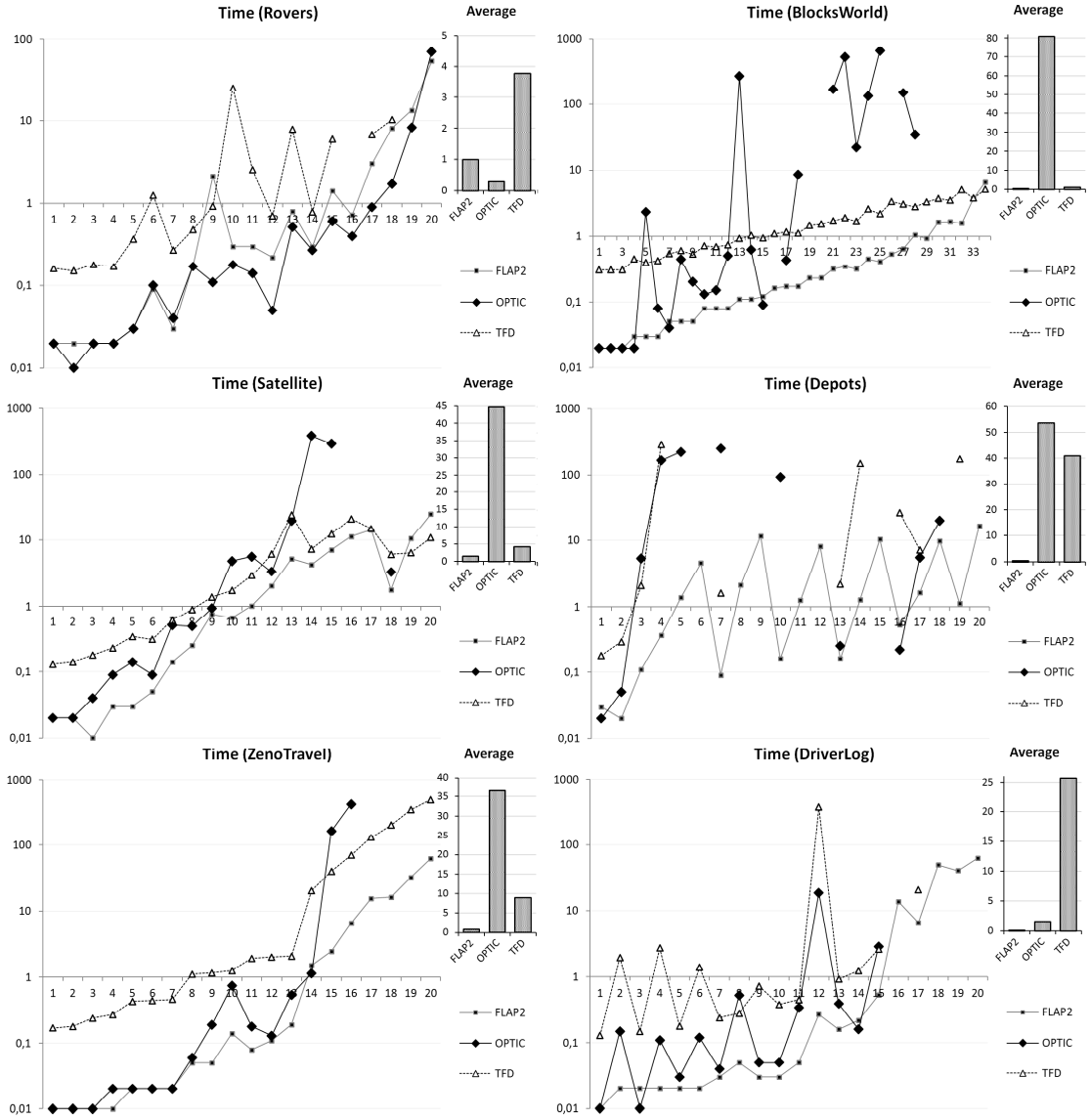


Figure 4 Planning time (in seconds) of FLAP2, OPTIC and TFD.

As a future extension, we intend to investigate the adaptation of the heuristic functions of FLAP2 to optimise the makespan and to mitigate the problem of h_{DTG} with dead-end states in non-reversible domains. Then, we want to exploit the good performance of FLAP2 and its flexibility as a partial-order planner to develop a new version for dealing with temporal planning problems.

Acknowledgements

This work has been partially supported by project FEDER-MINECO TIN2014-55637-C2-2-R.

References

Benton, J., Coles, A., and Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 2–10.

Chen, Y., Wah, B., and Hsu, C. (2006). Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, 26:323–369.

Coles, A., Coles, A., Fox, M., and Long, D. (2010). Forward-chaining partial-order planning. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 42–49.

Domshlak, C., Karpas, E., and Markovitch, S. (2010). To max or not to max: Online learning for speeding up optimal planning. *AAAI Conference on Artificial Intelligence*.

Eyerich, P. (2012). Preferring properly: Increasing coverage while maintaining quality in anytime temporal planning. *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 312–317.

Eyerich, P., Mattmüller, R., and Röger, G. (2009). Using the context-enhanced additive heuristic for temporal and numeric planning. *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Articial Intelligence Research*, 26:191–246.

Helmert, M., Haslum, P., Hoffmann, J., and Nissim, R. (2013). Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*.

Helmert, M., Röger, G., and Karpas, E. (2011). Fast Downward Stone Soup: A baseline for building planner portfolios. *Proceedings of the ICAPS-2011 Workshop on Planning and Learning (PAL)*, pages 28–35.

Hoffman, J. and Nebel, B. (2001). The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Hoffmann, J. (2002). Extending FF to numerical state variables. *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, pages 571–575.

Hoffmann, J., Porteous, J., and Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278.

Keyder, E. and Geffner, H. (2008). Heuristics for planning with action costs revisited. *European Conference on Artificial Intelligence (ECAI)*, pages 588–592.

Khouadjia, M., Schoenauer, M., Vidal, V., Dréo, J., and Savéant, P. (2013). Multi-objective AI planning: Comparing aggregation and pareto approaches. *Proceedings of the 13th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, 7832:202–213.

Kvarnström, J. (2011). Planning for loosely coupled agents using partial order forward-chaining. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 138–145.

Lipovetzky, N. and Geffner, H. (2009). Inference and decomposition in planning using causal consistent chains. *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 217–224.

Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 29(1):127–177.

Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Sapena, O., Onainda, E., and Torreño, A. (2013). Forward-chaining planning with a flexible least-commitment strategy. *Congreso Internacional de la Asociacin Catalana de Inteligencia Artificial (CCIA)*, 256:41–50.

Sebastia, L., Onaindía, E., and Marzal, E. (2006). Decomposition of planning problems. *AI Communications*, 19(1):49–81.

Torreño, A., Onaindía, E., and Sapena, O. (2012). An approach to multi-agent planning with incomplete information. *European Conference on Artificial Intelligence (ECAI)*, 242:762–767.

Vidal, V. (2003). A lookahead strategy for solving large planning problems. *Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI)*, pages 1524–1525.

Vidal, V. (2011). YAHSP2: Keep it simple, stupid. *Proceedings of the 7th International Planning Competition (IPC-2011)*, pages 83–90.