A lookahead strategy for solving large planning problems*

Vincent Vidal
CRIL - Université d'Artois
rue de l'Université - SP 16
62307 Lens Cedex, France
vidal@cril.univ-artois.fr

Abstract

Relaxed plans are used in the heuristic search planner FF for computing a numerical heuristic and extracting helpful actions. We present a novel way for extracting information from the relaxed plan and for dealing with helpful actions, by considering the high quality of the relaxed plans. In numerous domains, the performance of heuristic search planning and the size of the problems that can be handled have been drastically improved.

1 Computing and using lookahead states

In classical forward state-space search algorithms, a node in the search graph represents a planning state and an arc starting from that node represents the application of one action to this state, that leads to a new state. In order to ensure completeness, all actions that can be applied to one state must be considered. The order in which these states will then be considered for development depends on the overall search strategy: depth-first, breadth-first, best-first...

Let us now imagine that for each evaluated state S, we knew a valid plan P that could be applied to S and would lead to a state closer to the goal than the direct descendants of S. It could then be interesting to apply P to S, and use the resulting state S' as a new node in the search. This state could be simply considered as a new descendant of S.

We have then two kinds of arcs in the search graph: the ones that come from the direct application of an action to a state, and the ones that come from the application of a valid plan to a state S and lead to a state S' reachable from S. We will call such states *lookahead states*, as they are computed by the application of a plan to a node S but are considered in the search tree as direct descendants of S. Nodes created for lookahead states will be called *lookahead nodes*. Plans labeling arcs that lead to lookahead nodes will be called *lookahead plans*. Once a goal state is found, the solution plan is then the concatenation of single actions for arcs leading to classical nodes and lookahead plans for arcs leading to lookahead nodes.

*This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TACT Programme.

The determination of an heuristic value for each state as performed in the FF planner [Hoffmann and Nebel, 2001] offers a way to compute such lookahead plans. FF creates a planning graph [Blum and Furst, 1997] for each encountered state S, using the relaxed problem obtained by ignoring deletes of actions and using S as initial state. A relaxed plan is then extracted in polynomial time and space from this planning graph. The length in number of actions of the relaxed plan corresponds to the heuristic evaluation of the state for which it is calculated. Generally, the relaxed plan for a state S is not valid for S, as deletes of actions are ignored during its computation. In numerous benchmark domains, we can observe that relaxed plans have a very good quality because they contain a lot of actions that belong to solution plans. We propose a way of computing lookahead plans from these relaxed plans, by trying as most actions as possible from them and keeping the ones that can be collected into a valid plan.

The lookahead algorithm and the modifications to the search algorithm are the following (all details can be found in [Vidal, 2002]). Each time a state S is evaluated, it is entered into the open list. The relaxed plan extracted by the evaluation function is used to compute a lookahead plan P which leads to a state S' reachable from S. If P is more than one action long, S' is evaluated and added to the open list. Let P be a relaxed plan for a state S. A lookahead plan P' is computed as follows: all actions of P are observed in turn. When an action a is applicable to S, it is added to the end of P' and S is updated (by the application of a). When all actions of P have been tried, this process is iterated without the actions that have been applied, until no action can be used.

Completeness and correctness of search algorithms are preserved by this process, because no information is lost: all actions that can be applied to a state are still considered, and because the nodes that are added by lookahead plans are reachable from the states they are connected to. The only modification is the addition of new nodes, corresponding to states that can be reached from the initial state.

2 Using helpful actions: the "optimistic" best-first search algorithm

In classical search algorithms, all actions that can be applied to a node are considered the same way: the states that they lead to are evaluated by an heuristic function and are then or-

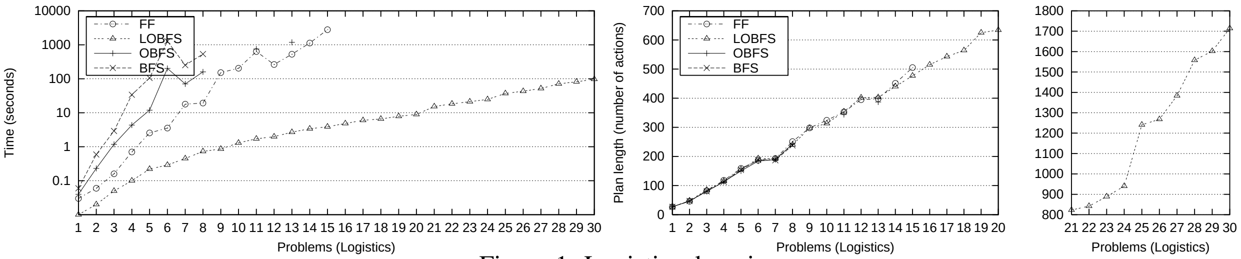

Figure 1: Logistics domain

	Rovers 24			Rovers 30	Driver 15			Driver 21		Driver 30
Init atoms	5920			35791	227			607		2130
Goals	33			127	10			38		163
	FF	OBFS	LOBFS	LOBFS	FF	OBFS	LOBFS	FF	LOBFS	LOBFS
Plan length	130	133	145	560	44	44	54	184	193	1574
Evaluated nodes	3876	2114	9	24	161	273	4	3266	8	38
Search time	418.32	430.95	1.97	44.35	0.21	0.84	0.02	207.89	0.45	93.92
Total time	422.48	437.92	8.87	219.13	0.26	0.97	0.14	209.40	1.96	284.65
	Satellite 21			Satellite 30	Logistics 13			Logistics 15		Logistics 30
Init atoms	971			10374	320			364		1140
Goals	124			768	65			75		200
	FF	OBFS	LOBFS	LOBFS	FF	OBFS	LOBFS	FF	LOBFS	LOBFS
Plan length	140	125	151	2058	398	387	403	505	477	1714
Evaluated nodes	22385	20370	5	5	16456	16456	4	45785	4	5
Search time	188.69	328.42	0.12	33.73	527.21	1181.95	0.29	2792.51	0.42	16.64
Total time	188.82	328.70	0.40	94.24	528.10	1184.35	2.68	2793.82	3.88	96.69

Table 1: Largest problems solved by OBFS, LOBFS and FF

dered, but there is no notion of preference over the actions themselves. Such a notion of preference during search has been introduced in the FF planner, with the concept of *helpful actions*. Once a relaxed plan is extracted for a state S, the actions of the relaxed plan that are executable in S are considered as *helpful*, while the other actions are forgotten by the local search algorithm of FF. This strategy appeared to be too restrictive, so the set of helpful actions is augmented in FF by all actions executable in S that produce fluents that were considered as subgoals at the first level of the planning graph, during the extraction of the relaxed plan. The main drawback of this strategy, as used in FF, is that it does not preserve completeness: the actions executable in a state S that are not considered as helpful are simply lost. FF switches to a complete best-first algorithm when no solution is found.

We present a way to use such a notion of helpful actions in complete search algorithms, that we call *optimistic search algorithms* because they give a maximum trust to the informations returned by the computation of the heuristic. The principles are the following:

- Several classes of actions are created. In our implementation, we only use two of them: *helpful actions* (the restricted ones), and *rescue actions* that are all the actions that are not helpful.
- When a newly created state S is evaluated, the heuristic function returns the numerical estimation of the state and also the actions executable in S partitioned into their different classes. For each class, one node is created for the state S, that contains the actions of that class.
- Nodes containing helpful actions are always preferred for development over nodes containing rescue actions, whatever their numerical heuristic values are.

No information is lost by this process. The way nodes are developed is simply modified: a state S is developed first with helpful actions, some other nodes are developed, and then S can potentially be developed with rescue actions. As the union of helpful actions and rescue actions is equal to the set of all the actions that can be applied to S, completeness and correctness are preserved.

3 Experimental evaluation

We compare four planners: FF v2.3, and three different settings of our planning system called YAHSP (which stands for Yet Another Heuristic Search Planner[1]) implemented in Objective Caml: BFS (Best First Search: classical WA^* search, with $W = 3$. The heuristic is based on the computation of a relaxed plan as in FF), OBFS (Optimistic Best First Search: identical to BFS, with preference of helpful actions over rescue actions), and LOBFS (Lookahead Optimistic Best First Search: identical to OBFS, with lookahead states).

We report here complete results for Logistics domain (see Figure 1) and show in Table 1 some data about the largest problems solved by FF, OBFS and LOBFS, in order to realize the progress accomplished in the size of the problems that can be solved by a STRIPS planner, for five different domains.

Acknowledgments

Thanks a lot to Pierre Régnier for his help…

References

[Blum and Furst, 1997] A. Blum and M. Furst. Fast planning through planning-graphs analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Vidal, 2002] V. Vidal. A lookahead strategy for heuristic search planning. Technical Report 2002-35-R, IRIT, Université Paul Sabatier, Toulouse, France, 2002.

[1] http://www.cril.univ-artois.fr/∼vidal/yahsp.html