A Planning Graph Heuristic for Forward-Chaining Adversarial Planning

Pascal Bercher and **Robert Mattmüller** [1]

Abstract. In contrast to classical planning, in adversarial planning, the planning agent has to face an adversary trying to prevent him from reaching his goals. In this paper, we investigate a forward-chaining approach to adversarial planning based on the AO* algorithm. The exploration of the underlying AND/OR graph is guided by a heuristic evaluation function, inspired by the relaxed planning graph heuristic used in the FF planner. Unlike FF, our heuristic uses an adversarial planning graph with distinct proposition and action layers for the protagonist and antagonist. First results suggest that in certain planning domains, our approach yields results competitive with the state of the art.

1 Introduction

In many planning problems, the environment in which the agent acts is not static. The exogenous dynamics can be caused by "nature" or by one or more other agents sharing the same environment. Other agents can behave neutrally (simply following their own independent agenda or otherwise acting unpredictably), adversarially, or cooperatively with respect to the protagonist's goals. Here, we focus on adversarial problems. We assume complete observability, i.e., a plan will be a mapping from physical states to applicable actions. A usual approach to conditional (adversarial) planning is planning as model checking [5], whereas planning as heuristic search [3] tends to yield best results for static, deterministic problems. Both approaches are also used in general game playing [7]. Related work includes the dynamic programming approach by Hansen and Zilberstein [8], and, for partially observable problems, heuristic search in the belief space as implemented in the POND planner by Bryce et al. [4].

2 Adversarial Planning

We consider discrete adversarial planning problems under full observability with alternating turns. More formally, similar to STRIPS problems [6], an *adversarial planning problem* is given by a set of states $S = 2^P$ over a finite set of propositions P, an *initial state* $I \subseteq P$, two finite sets of *operators* O_p and O_a (controlled by the protagonist p and antagonist a, respectively), and a *goal condition* $G \subseteq P$. Operators have the form $o = \langle pre, add, del \rangle$, where $pre \subseteq P$ is the *precondition* and $add, del \subseteq P$ are the *add* and *delete lists* of o. An operator o is *applicable* in a state $s \subseteq P$ iff $pre \subseteq s$, and if applied, leads to the successor state $s' = (s \setminus del) \cup add$. A state s is a *goal state* iff $G \subseteq s$. The players take alternating turns, starting with the protagonist controlling O_p. We assume that the player to move is known in each state. The protagonist tries to reach a goal state in a finite number of steps, whereas the antagonist tries to prevent him from doing so. A *winning strategy* for the protagonist is a function mapping states in which he is to move to applicable operators, such that, against each possible strategy of the antagonist, a goal state will be reached in a finite number of steps.

Such an adversarial planning problem naturally corresponds to the problem of evaluating an AND/OR graph over the state space. OR (AND) nodes correspond to states where the protagonist (antagonist) is to move and arcs correspond to operator applications. The relevant part of a winning strategy for the protagonist corresponds to an acyclic subgraph containing (a) the initial state, (b) for each contained non-goal AND node all outgoing arcs and their target nodes, (c) for each contained non-goal OR node exactly one outgoing arc and its target node, and no further nodes or arcs, such that all leaf nodes are goal states.

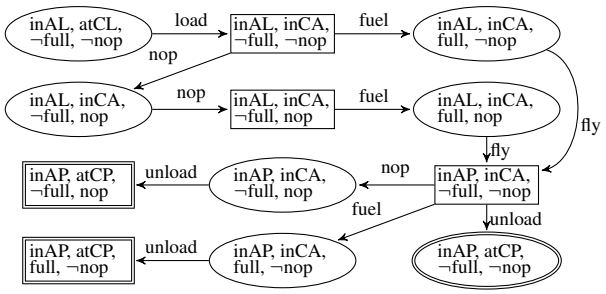

Figure 1. Cargo transport from London to Paris. The initial state is depicted on the upper left hand side, goal states are doubly framed. The protagonist moves in elliptic, the antagonist in rectangular nodes.

Consider for example a modified version of the Simple Rocket domain [2] with one airplane/rocket whose tank can be either full or empty, a set of cities, and a set of cargo packages which can be loaded and unloaded. Possible actions are *flying* from one city to another one if the tank is full, *loading* a package into the plane, *unloading* a package from the plane unless the same package has just been loaded without an intermittent flying action, *fueling* the plane if necessary, and performing *no-op*s. Flying and loading can only be done by the protagonist, fueling only by the antagonist, and unloading and no-ops by both, with the antagonist being barred from two consecutive no-ops without a flight in between. The goal of the protagonist is to transport the packages to specified target cities. The agents take turns, starting with the protagonist.

Assume two cities Paris and London, one package to be trans-

[1] University of Freiburg, Germany, {bercherp,mattmuel}@informatik.uni-freiburg.de. This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

		Breadth-First Search			AO* with FF heuristic			AO* with adversarial FF heuristic			MBP	
cities	pack's	time	mem	nodes	time	mem	nodes	time	mem	nodes	time	BDD nodes
2	1	0.014	1	44	0.025	1	37	0.026	1	37	0.000	6601
2	2	0.048	2	152	0.071	1	88	0.072	1	78	0.016	84424
3	3	0.354	6	2106	0.202	6	625	0.260	7	628	0.380	23068
3	4	0.870	49	8211	0.463	28	1871	0.232	17	605	1.780	165718
3	5	5.556	159	43785	1.437	98	6917	0.321	23	794	9.041	365272
3	6	87.691	987	237264	16.323	397	63498	1.157	25	4164	44.287	546666
4	6	—	3098	722750	76.718	698	169349	82.701	642	194304	130.064	834704
4	7	—	2192	771629	373.553	1840	510738	99.639	1487	225544	—	—
4	8	—	3889	912816	—	3356	738520	—	5440	914602	—	—

Figure 2. Experimental results for the transportation benchmark problems. We used a Java implementation, running on a machine with two Quad Xeon processors, 2.66 GHz, and a memory limitation of 16 GB RAM. The time-out, indicated by dashes, was set to ten minutes. Times are given in seconds, memory usage in MB. Memory usage and node counts in case of time-outs are the current values when the time-out occurred.

ported from London (atCL) to Paris (atCP), and the plane initially in London (inAL) with its tank empty (¬full). The variable "nop" is true iff the adversary has already performed a no-op since the last flight. A winning strategy for the protagonist is depicted in Figure 1.

3 Search Algorithm and Heuristic

As search algorithm, we used AO* [10] with maximization of cost estimates at AND nodes. The performance of the AO* algorithm depends on the choice of the evaluation function applied to the fringe nodes. To compute this function, we used an adaption of the graphplan-based [2] distance heuristic used in the FF planning system [9]. Just like the heuristic of the FF planning system, to which we will refer as *FF heuristic*, the *adversarial FF heuristic* uses relaxed operators, which we get by ignoring delete lists. For each agent $ag \in \{p, a\}$, let O^+_{ag} be the set of relaxed operators he controls. Fig. 3 shows the pseudocode of the adversarial FF heuristic. Lines 1 to 3 are equal to the forward step of the FF heuristic, except that there is not only one set of relaxed operators, but two distinct sets O^+_{ag} that belong to the two agents ag. Lines 4 to 11 correspond to the backward step of the FF heuristic. In addition, in line 12, the selected operators are put in two distinct sets SO^+_{ag}, one for each agent. After these two sets have been completely computed, in line 13 the value of the adversarial FF heuristic is calculated as follows: Since both agents move in turn, the number of moves needed to execute the plan is at most twice the number of operators contained in the larger one of the sets SO^+_{ag}, which we call SO^+_{max}. First, we calculate how many operators *have* to be applied by agent $max \in \{p, a\}$, which is $r := |SO^+_{max}| - |SO^+_{max} \cap O^+_{\overline{max}}|$, where $O^+_{\overline{max}}$ is the set of relaxed operators agent $\overline{max} \in \{p, a\} \backslash \{max\}$ controls. The value of the heuristic can then be calculated as $\max\{2r, |SO^+_p| + |SO^+_a|\}$.

4 Experimental Results

We experimented with solvable problems from the example domain described above with varying numbers of cities and packages. We compared running times, memory usage and node creations for uninformed breadth-first search, AO* search with the FF heuristic under the assumption of full cooperation, and AO* search with the adversarial FF heuristic. In addition, we encoded the same tasks as conditional planning problems under full observability in NuPDDL and solved them using MBP [5]. The results are summarized in Fig. 2.

5 Conclusion

The results in Fig. 2 suggest that in domains where the antagonist controls operators that may contribute to a plan, AO* search with adversarial FF heuristic often outperforms AO* search with FF heuristic and uninformed search. It is competitive with the symbolic approach used in MBP.

```
1  while G is not contained in the current layer i do
2      Let S[i] be the set of all state variables in layer i.
3      Let O+[i] be the set of all relaxed operators that are applicable in
         layer i and that belong to agent ag ∈ {p, a}, who is to move in
         layer i. Increment i.
4  Let G[m] be G.
5  for layer j := m − 1 to 0 do
6      foreach state variable g ∈ G[j + 1] do
7          if g ∈ S[j] then
8              Put g into G[j].
9          else
10             Put a relaxed operator o+ into SO+[j] that is in O+[j]
                 and that creates g.
11             Put the precondition pre of o+ into G[j].
12     Put all selected operators of SO+[j] into SO+_ag, the set of all
         selected operators of agent ag ∈ {p, a} who is to move in layer j.
13 If possible, shift operators from SO+_p to SO+_a (or vice versa) to ensure
     that the difference between |SO+_p| and |SO+_a| is as small as possible.
     Calculate and return the least number of moves that will be needed to
     apply all operators of the two rearranged sets.
```

Figure 3. Adversarial FF heuristic.

REFERENCES

[1] Pascal Bercher and Robert Mattmüller, 'A Planning Graph Heuristic for Forward-Chaining Adversarial Planning', Technical Report 238, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, (2008).

[2] Avrim L. Blum and Merrick L. Furst, 'Fast Planning Through Planning Graph Analysis', in *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pp. 1636–1642, (1995).

[3] Blai Bonet and Héctor Geffner, 'Planning as Heuristic Search', *Artificial Intelligence*, **129**(1-2), 5–33, (2001).

[4] Daniel Bryce, Subbarao Kambhampati, and David E. Smith, 'Planning Graph Heuristics for Belief Space Search', *Journal of Artificial Intelligence Research*, **26**, 35–99, (2006).

[5] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso, 'Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking', *Artificial Intelligence*, **147**(1–2), 35–84, (2003).

[6] Richard E. Fikes and Nils J. Nilsson, 'STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving', *Artificial Intelligence*, **2**(3–4), 189–208, (1971).

[7] Michael R. Genesereth, Nathaniel Love, and Barney Pell, 'General Game Playing: Overview of the AAAI Competition', *AI Magazine*, **26**(2), 62–72, (2005).

[8] Eric A. Hansen and Shlomo Zilberstein, 'Heuristic Search in Cyclic AND/OR Graphs', in *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pp. 412–418, (1998).

[9] Jörg Hoffmann and Bernhard Nebel, 'The FF Planning System: Fast Plan Generation Through Heuristic Search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[10] Nils J. Nilsson, *Principles of Artificial Intelligence*, Springer, 1980.