Parallelizing State Space Plans Online

Romeo Sanchez Nigenda & Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
Email: $\{$rsanchez,rao$\}$@asu.edu

Abstract

Searching for parallel solutions in state space planners is a challenging problem, because it would require the planners to branch on all possible subsets of parallel actions, exponentially increasing their branching factor. We introduce a variant of our heuristic state search planner *AltAlt*, which generates parallel plans by using greedy online parallelization of partial plans. Empirical results show that our online approach outperforms post-processing (offline) techniques in terms of the quality of the solutions returned.

1 Introduction

Parallel plans allow concurrent execution of multiple actions at each time step. While disjunctive planners such as Graphplan [Blum and Furst, 1997], SATPLAN [Kautz and Selman, 1996] and GP-CSP [Do and Kambhampati, 2000] seem to have no trouble generating such parallel plans, planners that search in the space of states are overwhelmed by this task [Haslum and Geffner, 2000]. The problem for state space planners is the exponential branching factor of the search space. One approach to solve this problem is to post-process the sequential plans generated by the state space planners. This could easily be done–using approaches such as those explored by Backstrom [Backstrom, 1998], but they are limited to transforming the sequential plan given as input. An alternative that we introduce in this article, involves incremental online parallelization. Specifically, our planner *AltAlt*^{p} [Sanchez and Kambhampati, 2003], which is a variant of the *AltAlt* planner [Sanchez et al., 2000; Nguyen et al., 2000], attempts to parallelize a heuristically selected search branch with other independent actions, and then tries to rearrange the evolving parallel plan using a plan-compression algorithm.

We will provide a brief background on *AltAlt* planning system on Section 2. Section 3 describes the generation of parallel plans in *AltAlt*^{p}. Section 4 presents some empirical evaluation of *AltAlt*^{p}. Finally, Section 5 summarizes our contributions.

2 *AltAlt* Background

The *AltAlt* planning system is based on a combination of Graphplan [Blum and Furst, 1997] and heuristic state space search [Haslum and Geffner, 2000] technology. The problem specification and the action template description are first fed to a Graphplan-style planner, which constructs a planning graph for that problem in polynomial time. This planning graph structure is then fed to a heuristic extractor module that is capable of extracting a variety of effective heuristics [Nguyen et al., 2000]. These heuristics, along with the problem specification, and the set of ground actions in the final action level of the planning graph are then fed to a regression state-search planner.

3 Plan Compression Algorithm

AltAlt^p considers at each step during search only those (pairwise parallel) actions that individually regress to states with lower heuristic estimates than their parent node to parallelize (fatten) a search branch. This greedy nature of the fattening procedure while useful in avoiding the addition of irrelevant actions to the plan, may also sometimes preclude actions that are ultimately relevant. When this happens, the parallel length of the solution plan is likely to be worsened, as more steps may be needed to support the preconditions of such actions. In order to offset this negative effect of greediness, *AltAlt*^{p} re-arranges the partial plan to promote such actions higher up the search branch using a plan-compression algorithm (See Figure 1). Specifically, before expanding a given node S, *AltAlt*^{p} checks to see if any of the actions in A_s leading to S from its parent node, can be pushed up to higher levels in the search branch. The push-up procedure is called each time before a node gets expanded, and it will try to compress the current partial plan. For each of the actions $a \in A_s$ we find the highest ancestor node S_x of S in the search branch to which the action can be applied (i.e., it gives some literal in S_x without deleting any other literals in S_x, and it is pairwise independent of all the actions currently leading out of S_x). Once S_x is found, a is then removed from the set of actions leading to S and introduced into the set of actions leading out of S_x (to its child in the current search branch). Next, the states in the search branch below S_x are adjusted to reflect this change. The adjustment involves recomputing the regressions of all the search nodes below S_x. At first glance,

```
pushUP(S)
    $A_s \leftarrow$ get actions leading to S
    forall $a \in A_s$
        $x \leftarrow 0$
        $S_x \leftarrow$ get parent node of S
        Loop
            $A_x \leftarrow$ get actions leading to $S_x$
            If $(parallel(a, A_x))$
                $x \leftarrow x + 1$
                $S_x \leftarrow$ get parent node of $S_{x-1}$
            Else
                $A_{x-1} \leftarrow A_{x-1} + a$
                $A_s \leftarrow A_s - a$
        End Loop
                /**Adjusting the partial plan
    $S_x \leftarrow$ get highest ancestor x in history
    $createNewBranchFrom(S_x)$
    while $x > 0$
        $S_{new} \leftarrow$ regress $S_x$ with $A_{x-1}$
        $S_x \leftarrow S_{new}$
        $x \leftarrow x - 1$
END;
```

Figure 1: Push Up Procedure

this might seem like a transformation of questionable utility since the preconditions of a (and their regressions) just become part of the descendants of S_x, and this does not necessarily reduce the length of the plan. We however expect a length reduction because actions supporting the preconditions of a will get "pushed up" eventually during later expansions.

4 Empirical Evaluation

We compare our online parallelization approach to post-processing, we have implemented Backstrom's 'Minimal De-ordering Algorithm" from [Backstrom, 1998], and used it to post-process the sequential plans produced by *AltAlt*. We can see on Figure 2 some empirical results on the Zeno domain [Long and Fox, 2002].

As expected, the original *AltAlt* has the longest plans since it allows only one action per time step. The plot shows that post-processing techniques do help in reducing the makespan of the plans generated by *AltAlt*. However, we also notice that *AltAlt^p* outputs plans with better makespan than either *AltAlt* or *AltAlt* followed by post-processing. This shows that online parallelization is a better approach than post-processing sequential plans. *AltAlt^p* also does not result in any additional run-time overhead with respect to *AltAlt*. Further experiments reported in [Sanchez and Kambhampati, 2003] show that our approach is competitive or outperforms other state of the art approaches in generating parallel plans.

5 Conclusions

We have introduced an approach to generate parallel plans in the context of state space search. Our approach tries to avoid the branching factor blow up by greedy and online parallelization of the evolving partial plans. A plan compression procedure is used to offset the ill effects of the greedy search. Our empirical results show that our approach seems to return better quality plans than post-processing the sequential plans.

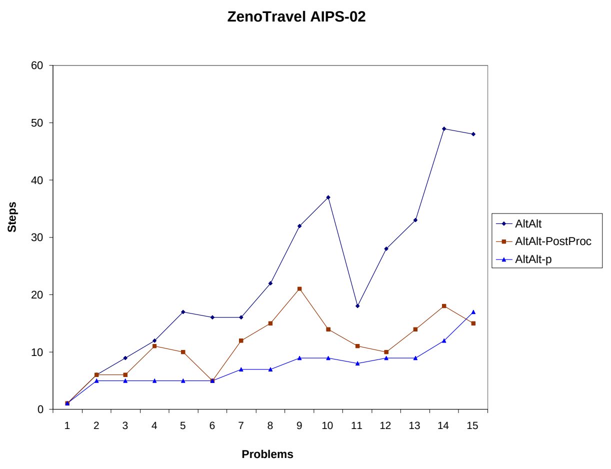

Figure 2: *AltAlt* and Post-Processing *vs AltAlt^p*

In future, we plan to adapt the *AltAlt^p* approach to metric temporal domains, where the need of concurrency is more pressing. It is in these scenarios where more powerful heuristics based on resource usage can be further investigated to improve the parallelism of state space planners.

References

[Blum and Furst, 1997] A. Blum and M.L. Furst. Fast planning through planning graph analysis. In *Artificial Intelligence*. 90(1-2). 1997.

[Backstrom, 1998] C. Backstrom. Computational Aspects of Reordering Plans In *Journal of Artificial Intelligence Research, 1998*.

[Long and Fox, 2002] D. Long and M. Fox. Results of the AIPS 2002 Planning Competition.
URL: *http://www.dur.ac.uk/d.p.long/competition.html*.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic and stochastic search. In *Proc. AAAI 1996*. Portland, OR.

[Do and Kambhampati, 2000] Minh B. Do, and S. Kambhampati. Solving Planning Graph by Compiling it into a CSP. In *Proc. AIPS 2000*

[Haslum and Geffner, 2000] P. Haslum and H. Geffner. Admissible Heuristics for Optimal Planning. In *Proc. AIPS 2000*.

[Sanchez et al., 2000] R. Sanchez, X. Nguyen and S. Kambhampati. AltAlt: Combining the advantages of Graphplan and Heuristics State Search. In*Proc. KBCS 2000*. Bombay, India.

[Sanchez and Kambhampati, 2003] R. Sanchez and S. Kambhampati. AltAltp: Online Parallelization of Plans with Heuristic State Search. To appear in *JAIR*.

[Nguyen et al., 2000] X. Nguyen, S. Kambhampati, and R. Sanchez Nigenda. Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. In *Journal of Artificial Intelligence, 2002*.