Complete Anytime Beam Search

Weixiong Zhang
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
Email: zhang@isi.edu

ABSTRACT

Beam search executes a search method, such as best-first search or depth-first search, but may abandon nonpromising search avenues in order to reduce complexity. Although it has existed for more than two decades and has been applied to many real-world problems, beam search still suffers from the drawback of possible termination with no solution or a solution of unsatisfactory quality. In this paper, we first propose a domain-independent heuristic for node pruning, and a method to reduce the possibility that beam search will fail. We then develop a complete beam search algorithm. The new algorithm can not only find an optimal solution, but can also reach better solutions sooner than its underlying search method. We apply complete beam search to the maximum boolean satisfiability and the symmetric and asymmetric Traveling Salesman Problems. Our experimental results show that the domain-independent pruning heuristic is effective and the new algorithm significantly improves the performance of its underlying search algorithm.

1 Introduction

Beam search [2] executes a state-space search algorithm, such as best-first search or depth-first search [16], but may use heuristic pruning rules to discard nonpromising search alternatives that seem unlikely to lead to a solution or appear to lead to a solution of unsatisfactory quality. Heuristic pruning keeps the size of the *beam*, the remaining search alternatives, as small as possible, in order to possibly find a solution quickly. The idea of beam search is simple, and has been successfully applied to many different problems, such as learning [4], jobshop scheduling [5], speech recognition [12], planning [14], and vision [19].

Despite the fact that beam search has existed for more than two decades and has been applied to many real-world applications, it has not been carefully studied. Beam search is a heuristic technique used to reduce search complexity. It has, however, a serious drawback of possible termination with no solution or a solution of unsatisfactory quality. In other words, beam search is an *incomplete algorithm* that is not guaranteed to find a solution even if one exists. The pruning power and possibility of finding a solution depend on the accuracy of the pruning rules used. Effective heuristic rules are generally problem dependent, and their effectiveness comes from deep understanding of the problem domains. In practice, it is difficult to find effective heuristic rules that can strike the right balance of finding a desired goal and using the minimal amount of computation.

In this paper, we first propose a domain-independent heuristic pruning rule that uses only heuristic node evaluations (Section 3), and a method to increase the possibility that beam search finds a solution (Section 4). We then develop a complete anytime beam search algorithm that is able to continuously find better solutions and eventually reach an optimal solution (Section 5). We apply complete anytime beam search to the maximum boolean satisfiability and the symmetric and asymmetric Traveling Salesman Problems, and investigate the anytime feature of the new algorithm on these real problems (Section 6). We discuss related work in Section 7, and finally conclude in Section 8.

2 Beam Search

Beam search runs a state-space search method, such as best-first search (BFS) or depth-first search (DFS). What sets beam search apart from its underlying search is the use of heuristic rules to prune search alternatives before exploring them. Note that these heuristic pruning rules are different from the pruning rule based on monotonic node costs and an upper bound α on the cost of an optimal goal. To make the paper self contained, we list beam search in Table 1, where R represents the set of heuristic pruning rules, and $c(n)$ is the cost of n.

Beam search can use any strategy, such as best first or depth first, to select a node at line 3 of Table 1. The

Table 1: Beam search.

```
BeamSearch(problem, R, α)
1.  open ← {problem};
2.  WHILE (open ≠ ∅)
3.    n ← a node in open;
4.    If (n is a desired goal) exit;
5.    If (n is a goal & c(n) < α) α ← c(n);
6.    Remove n from open;
7.    Generate all children of n;
8.    Discard a child n' if c(n') ≥ α;
9.    Discard n' if pruned by a rule in R;
10.   Insert remaining children into open.
```

only difference between beam search and its underlying search method is line 9 of Table 1, where heuristic pruning rules are used to prune nodes. The algorithm terminates in two cases. It may stop when a required goal node is reached at line 4. It may also terminate when no node is left for further exploration, which is a failure if a goal exists. This failure makes beam search *incomplete*, an unfavorable feature for practical use.

3 Domain-Independent Pruning Rule

Whether or not beam search can find a solution depends on the heuristic pruning rules it uses. Accurate heuristic rules can provide strong pruning power by eliminating the nodes that do not need to be explored. On the other hand, effective heuristic pruning rules are generally domain dependent and require a deep understanding of problem domains.

In this research, we are interested in domain-independent heuristic pruning rules. We propose one such pruning rule in this section. It uses information of static heuristic evaluations of the nodes in a state space. Without loss of generality, we assume that heuristic node evaluation function is monotonically nondecreasing with node depth.

Consider a node n and its child node n' in a state space. Let $c(n)$ be the static heuristic evaluation or static node cost of n. If $c(n')$ exceeds $c(n)$ by δ, i.e., $c(n') > c(n) + \delta$, then n' is discarded, where δ is a predefined, positive parameter. Thus, a smaller δ represents a stronger pruning rule. This heuristic pruning rule was derived from the intuition that if a child node has a significantly large cost increase from its parent, it is more likely that the child may lead to a goal node with a large cost. The effectiveness of this rule will be examined in Section 6.

4 Reducing Failure Rate

The use of heuristic pruning rules in beam search is a two-edged sword. On one side, it reduces the amount of search, solving some problems in reasonable time. On the other side, however, it runs the risk of missing a solution completely, making it incomplete and inapplicable to some applications. When beam search fails to find a solution, its heuristic pruning rules abandon too many search alternatives. The pruned nodes are deadend nodes, which do not have children, as far as beam search is concerned.

The idea of increasing the possibility of reaching a goal node is to reduce the possibility of creating deadends. When a node has children, the pruning may be too strong if all its child nodes are discarded. Instead of abandoning all children based on the pruning rules, keeping at least one child will prevent treating the current node as a deadend. We call this a modification rule, a meta rule that governs how the pruning rules should be used. To find a high-quality goal node, the modification rule chooses to explore the child node that has the minimum cost among all the children.

Table 2: Complete beam search algorithm.

```
CompleteBeamSearch(problem, R, α)
  DO
    Call BeamSearch(problem, R, α);
    Weaken heuristic pruning rules in R;
  WHILE (no desired goal found & a rule ∈ R
         applied in the last iteration)
```

5 Complete Anytime Beam Search

We have developed a *complete beam search algorithm* (CBS) using a technique called *iterative weakening* [18]. The algorithm is simple and straightforward. It iteratively runs a series of beam searches using weaker heuristic pruning rules in subsequent iterations.

5.1 The algorithm

Specifically, the algorithms runs as follows. It first runs the original beam search. If a desired solution is found, the algorithm terminates. Otherwise, the heuristic pruning rules are weakened, and the algorithm runs another iteration of beam search. This iterative process continues until a required solution has been found, or no heuristic pruning rule was applied in the last iteration. The latter case means that either no required solution exists, or the solution found so far is optimal, because the last iteration runs the underlying BFS or DFS (which is guaranteed to find an optimal solution). This case also means that this algorithm is complete. Table 2 lists the complete beam search algorithm.

When domain-specific pruning rules are used, we can reduce the number of pruning criteria or remove individual components in pruning rules to reduce their

pruning power. For the domain-independent pruning rule suggested in Section 3, we can increase the parameter δ to make the rule weak. Recall that a child node n' will be pruned if its cost $c(n')$ is greater than the cost of its parent $c(n)$ by δ. By increasing δ, we reduce the possibility of pruning the child node.

5.2 Anytime feature

Anytime algorithms [3] are an important class of algorithms for real-time problem solving. An anytime algorithm first finds a solution quickly, and then successively improves the quality of the best solution found so far, as long as more computation is available. CBS is an anytime algorithm for solving combinatorial optimization problems. The state spaces of combinatorial optimization problems, such as those considered in Section 6, have leaf nodes that are goal states. When solving these problems, CBS can find a solution quickly in an early iteration, and continuously find better solutions in subsequent iterations.

When using best-first search (BFS) as its underlying search method, CBS turns BFS into an anytime search algorithm. Note that BFS does not provide a solution until its termination [16]. By using heuristic pruning rules, an iteration of CBS searches a small portion of the state space that is examined by the full BFS, and may obtain a suboptimal solution quickly. With weaker pruning rules, a larger portion of the state space will be explored in order to find better solutions. By repeatedly using weaker pruning rules, CBS can incrementally find better solutions until it visits an optimal solution.

Now consider depth-first search (DFS). DFS is an anytime algorithm by nature [22], as it systematically explores leaf nodes of a state space. Based on our experiments on random search trees, CBS significantly improves DFS' anytime feature, finding better solutions sooner. The real challenge is whether complete beam search can outperform DFS on real problems. This question will be favorably answered in the next section.

6 Applications

In this section, we apply the complete beam search (CBS) algorithm and the domain-independent heuristic pruning rule of Section 3, plus the modification rule of Section 4, if necessary, to three NP-complete [15] combinatorial optimization problems, the maximal boolean satisfiability and the symmetric and asymmetric Traveling Salesman Problems. The purpose of this experimental study is to examine the anytime feature of CBS and the effectiveness of the domain-independent pruning rule. In our experiments, we use DFS as CBS' underlying search method, and compare CBS' performance profile [23] against that of DFS.

The performance profile of an algorithm characterizes the quality of its output as a function of computation time. Denote $prof(A,t)$ as the performance profile of algorithm A. We define $prof(A,t)$ as

$$prof(A,t) = 1 - error(A,t), \quad (1)$$

where $error(A,t)$ is the error of solution cost of A at time t relative to the optimal solution cost. During the execution of A, $prof(A,t) \leq 1$; and at the end of its execution, $prof(A,t) = 1$. In our experiments, we compare $prof(CBS,t)$ against $prof(DFS,t)$.

Our domain-independent heuristic pruning rule uses a parameter δ. In order to set the value of δ properly, we need information about how much the cost of a node will increase from that of its parent. It would be ideal if we knew the distribution of cost differences between nodes and their parents for a given application. Such information has to be collected or learned from the problems to be solved. This can be done in two ways. The first is offline sampling. If some sample problems from an application domain are available, we can first solve these problems using BFS or DFS, and use the costs of the nodes encountered during the search as samples to calculate an empirical distribution of node-cost differences. The second method is online sampling, which can be used when sample problems are not provided. Using this sampling method, the first iteration of CBS does not apply heuristic pruning rules until a certain number of nodes have been generated. In our experiments presented in the rest of this section, we use the online sampling method and DFS as CBS' underlying search method. In our implementation, CBS does not use pruning rules until it has reached the first leaf node. All the nodes generated in the process of reaching the first leaf node are used as initial samples for computing an empirical distribution.

Using the empirical distribution of node-cost differences, we can set parameter δ. What value of δ to use in the first iteration and how it is updated from one iteration to the next are critical factors that directly determine CBS' performance. In general, these factors must be dealt with on a case by case basis, depending on the applications. In our experiments on the following three combinatorial problems, the initial δ is set to a value δ_I such that a node-cost difference is less than δ_I with probability p equal to 0.1. The next iteration increases probability p by 0.1, and so forth, until the heuristic pruning rule has not been used in the latest iteration or probability p is greater than 1.

Due to space limitations, in the following discussions we will be brief on our implementations of DFS on the three problems.

6.1 Maximum boolean satisfiability

We are concerned with boolean 3-satisfiability (3-Sat), a special case of the constraint-satisfaction problem (CSP). A 3-Sat involves a set of boolean variables, and

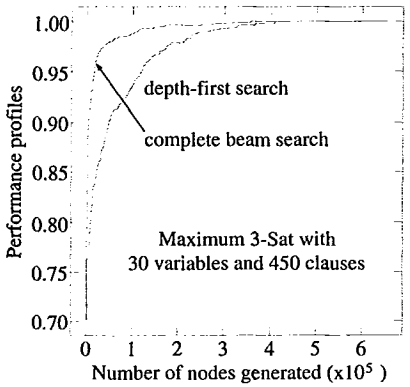

Figure 1: CBS vs. DFS on maximum 3-Satisfiability.

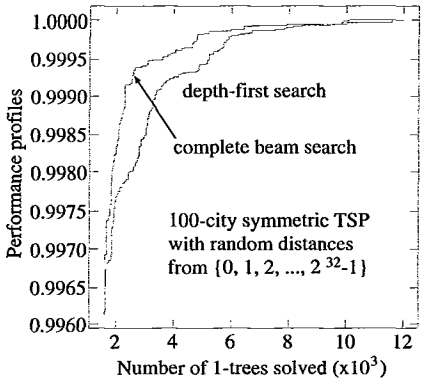

Figure 2: CBS vs. DFS on the symmetric TSP.

a set of disjunctive clauses of 3 literals (variables and their negations) which defines constraints of acceptable combinations of variables. There are many practical CSPs in which no value assignment can be found that does not violate a constraint; see [6] for discussion and references therein. In this case, one option is to find an assignment such that the total number of satisfied constraints is maximized. In our experiment, we consider maximum 3-Sat.

Maximum 3-Sat can be optimally solved by DFS as follows. The root of the search tree is the original problem with no variable specified. One variable is then chosen and set to either true or false, thus decomposing the original problem into two subproblems. Each subproblem is then simplified as follows. If the selected variable is set to true, a clause can be removed if it contains this variable. A clause can also be discarded if it contains the negation of a variable that is set to false. Furthermore, a variable can be deleted from a clause if the negation of the literal is set to true. Since the two values of a variable are mutually exclusive, so are the two subproblems generated. Therefore, the state space of the problem is a binary tree without duplicate nodes. The cost of a node is the total number of clauses violated, which is monotonically nondecreasing with the depth of the node. In our implementation of DFS, we use the *most occurrence heuristic* to choose a variable; in other words, we choose an unspecified variable that occurs most frequently in the set of clauses.

We generated maximum 3-Sat problem instances by randomly selecting three variables and negating them with probability 0.5 for each clause. Duplicate clauses were removed. The problem instances we used have a large ratio of the number of clauses to the number of variables (clause to variable ratio), since random 3-Sat problems with a small clause-to-variable ratio are generally satisfiable [13].

In our experiments, we studied the effects of the modification rule proposed in Section 4, which will explore the best child node if the two children of a node are pruned by the heuristic pruning rule. On random 3-Sat, CBS without the modification rule cannot compete with DFS. This is due to two factors. The first is the small branching factor 2 of the state space, so that deadend nodes can be easily generated if the modification rule is not applied. The second is that the initial value of δ is too small, thus most early iterations of CBS cannot find a solution at all. However, CBS with the modification rule is significantly superior to DFS.

Figure 1 shows the experimental result of CBS using the modification rule on 3-Sat with 30 variables and 450 clauses, averaged over 100 random problem instances. The vertical axis presents the performance profiles of CBS and DFS, and the horizontal axis is the number of nodes generated. The comparison has an almost identical picture when the horizontal axis is plotted in terms of CPU time on a SUN Sparc 2 machine, as the time spent for online sampling is negligible. Figure 1 shows that CBS significantly improves the anytime performance of DFS, finding better solutions sooner.

6.2 The symmetric TSP

Given n cities $\{1, 2, ..., n\}$ and a cost matrix $(c_{i,j})$ that defines a cost between each pair of cities, the *Traveling Salesman Problem* (TSP) is to find a minimum-cost tour that visits each city once and returns to the starting city. When the cost from city i to city j is the same as that from city j to city i, the problem is the *symmetric TSP* (STSP).

In our implementation of DFS, we use the Held-Karp lower bound function [9] to compute node costs. This function iteratively computes a Lagrangian relaxation on the STSP, with each step constructing a 1-tree. A 1-tree is a minimum spanning tree (MST) [15] on $n-1$ cities plus the two shortest edges from the city not in the MST to two cities in the MST. Note that a complete TSP tour is a 1-tree. If no complete TSP tour has been found after a predefined number of steps of Lagrangian relaxation, which is $n/2$ in our experiment, the problem

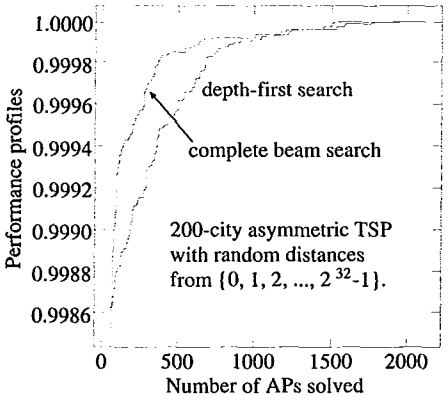

Figure 3: CBS vs. DFS on the asymmetric TSP.

is decomposed into at most three subproblems using the Volgenant and Jonker's branching rule [20]. Under this decomposition rule, the state space of the STSP is a tree without duplicate nodes.

We generated STSP problem instances by uniformly choosing a cost between two cities from $\{0, 1, 2, \cdots, 2^{32} - 1\}$. The experiment results show that CBS without the modification rule of Section 4 is substantially degenerated from DFS, due to the same reasons described in Section 6.1, on maximum 3-sat.

Figure 2 shows the experimental result of CBS with the modification rule on 100-city random STSPs, averaged over 100 problem instances. The vertical axis still presents the performance profiles of CBS and DFS. The horizontal axis is the numbers of 1-trees solved by both algorithms. We do not use the number of tree nodes generated as a time measure, because generating a node requires the solution of one to $n/2$ 1-trees based on Lagrangian relaxation, and the CPU time is proportional to the number of 1-trees solved. Figure 2 shows that CBS improves DFS on random STSPs.

6.3 The asymmetric TSP

When the cost from city i to city j is not necessarily equal to that from city j to city i, the TSP is the *asymmetric TSP* (ATSP). The most efficient approach known for optimally solving the ATSP is subtour elimination, with the solution to the assignment problem (AP) as a lower-bound function [1]. The AP is to assign to each city i another city j, with the cost from i to j as this assignment, such that the total cost of all assignments is minimized. An AP is a relaxation on the ATSP without the requirement of a complete tour. See [1] for the details of this algorithm. In short, the problem space of subtour elimination can be represented by a tree with maximum depth less than n^2. We used this algorithm in our experiments.

It is worth mentioning that the branching factor of an ATSP state space is large, proportional to the number of cities. Therefore, the modification rule of Section 4 does not matter much. In fact, the experimental result of CBS with the modification rule is slightly worse than that without the modification rule.

We used random ATSP in our experiments. The costs among cities were uniformly chosen from $\{0, 1, 2, \cdots, 2^{32} - 1\}$. Figure 2 shows the experimental results of CBS without the modification rule on 200-city random ATSP, averaged over 100 instances. Figure 3 shows that CBS outperforms DFS on random ATSPs.

7 Related Work and Discussions

A restricted version of beam search was defined in [21], which runs breadth-first search, but keeps a fixed number of nodes active on each depth. The definition used in this paper follows that in [2] and is more general. The most closely related work on beam search is the early applications of beam search to various problems which demonstrated its effectiveness [2, 4, 5, 12, 14, 19].

CBS is a combination of beam search heuristics and iterative weakening [18]. Iterative weakening directly follows iterative deepening [11] and iterative broadening [7]. They all repeatedly apply a search process, but with stronger or weaker parameters in different passes. It has been shown that a given set of search policies should be applied in an increasing order of the search complexities that these policies incur [18].

CBS bears a close similarity to iterative broadening. Briefly, iterative broadening first carries out a search with a breadth limit of two, and if it fails, the algorithm repeats the search with breadth limit of three, and so on, until it finds a solution. Early passes of both algorithms comb through the state space for better solutions, and they gradually extend the coverage of their exploration by increasing the search breadth. The difference between these two algorithms is that iterative broadening extends its search breadth in a predetermined fashion, while CBS broadens its search depending on how heuristic pruning rules are weakened. If we treat the way that search breadth is extended in a predefined way as a special heuristic, iterative broadening can then be considered as a special case of CBS.

CBS using the domain-independent pruning rule of Section 3 is symmetric to Epsilon search [17]. In Epsilon search, a node with a cost no greater than its parent's cost plus ϵ is treated as if it has the same cost as its parent, so as to force an early exploration of the node, while in CBS a node with cost greater than its parent's cost plus δ is considered as if it has an infinitely large cost, so as to postpone the exploration of the node.

Anytime algorithms are important tools for problem solving in a real-time setting and with resource constraints [3, 10, 23]. Although it is well known that

many search methods, such as depth-first search and local search, can be used as anytime algorithms, little work has been done to improve their anytime performance, except that of [8, 17]. In [8], non-admissible heuristic search runs as an anytime algorithm. CBS is more similar to Epsilon search [17], since both manipulate state space during their searches.

8 Conclusions

In this paper, we made beam search heuristics into a complete search algorithm, called complete beam search (CBS), by using the iterative weakening technique. CBS can not only find an optimal solution, but it also reaches better solutions sooner than its underlying search method. CBS is an anytime algorithm for solving combinatorial optimization problems. We also proposed a domain-independent node pruning heuristic. We applied CBS and the domain-independent heuristic pruning rule to three NP-complete optimization problems, the maximum boolean satisfiability and the symmetric and asymmetric Traveling Salesman Problems. Our experimental results show that the domain-independent pruning rule is effective, and CBS significantly improves the efficiency and anytime performance of its underlying depth-first search.

Acknowledgments

This research was partially funded by NSF Grant IRI-9619554. Thanks to the anonymous reviewers for their insightful comments and for bringing [18] to my attention, and to Sheila Coyazo for proof reading this paper.

References

[1] E. Balas and P. Toth. Branch and bound methods. In E.L. Lawler *et al.*, editor, *The Traveling Salesman Problem*, pages 361–401. John Wiley & Sons, Essex, 1985.

[2] R. Bisiani. Search, beam. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 1467–1468. Wiley-Interscience, 2nd edition, 1992.

[3] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. AAAI-88*, pages 49–54, St. Paul, MN, Aug. 1988.

[4] G. Dietterich and R. S. Michalski. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16:257–294, 1981.

[5] M. S. Fox. *Constraint Directed Search: A Case Study of Job-Shop Scheduling.* PhD thesis, Carnegie Mellon University, 1983.

[6] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[7] M. L. Ginsberg and W. D. Harvey. Iterative broadening. *Artificial Intelligence*, 55:367–383, 1992.

[8] E. A. Hansen, S. Zilberstein, and V. A. Danilchenko. Anytime heuristic search: First results. Technical Report CMPSCI 97-50, CSD, University of Massachusetts, Sept. 1997.

[9] M. Held and R. M. Karp. The Traveling Salesman Problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[10] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proc. of the 3rd Workshop on UAI*, 1987.

[11] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[12] K.-F. Lee. *Large-Vocabulary Speaker-Dependent Continuous Recognition: The Sphinx System.* PhD thesis, Carnegie Mellon University, 1988.

[13] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, pages 459–465, San Jose, CA, July 1992.

[14] N. Muscettola, S. F. Smith, G. Amiri, and D. Patak. Generating space telescope observation schedules. Technical Report CMU-RI-TR-89-28, Carnegie Mellon University, 1989.

[15] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[16] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[17] J. C. Pemberton and W. Zhang. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81:297–325, 1996.

[18] F. J. Provost. Iterative weakening: Optimal and near-optimal policies for the selection of search bias. In *Proc. AAAI-93*, pages 769–775, 1993.

[19] S. Rubin. *The ARGOS Image Understanding System.* PhD thesis, Carnegie Mellon University, 1978.

[20] T. Volgenant and R. Jonker. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operations Research*, 9:83–89, 1982.

[21] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, second edition, 1992.

[22] W. Zhang. Truncated branch-and-bound: A case study on the asymmetric TSP. In *Proc. AAAI 1993 Spring Symp.: AI and NP-Hard Problems*, pages 160–166, 1993.

[23] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82:181–213, 1996.