Solution of Constrained Optimization Problems in Limited Time

Lon-Chan Chu and Benjamin W. Wah

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801
{chu,wah}@manip.crhc.uiuc.edu

Abstract

In this paper, we study efficient search algorithms for solving in limited time constrained optimization problems with bounded search space. Due to the constraints imposed, feasible solutions are rare and are hard to find. We present a class of algorithms called band search that operate in a bounded amount of memory space and that can find feasible solutions efficiently. We verify our results by solving asymmetric traveling salesperson and maze problems.

Keywords and Phrases: Band search, real-time search, constrained optimization problems.

1: Introduction

In this paper, we develop efficient algorithms for solving in limited time constrained combinatorial optimization problems with bounded search space. Due to the constraints imposed on these problems, feasible solutions are hard to find (which means that they cannot be found in polynomial time). We consider minimization problems only, as maximization problems are duals. For constrained minimization problems, a search node is characterized by a lower-bound value but not an upper-bound value, as algorithms for computing the upper-bound value generally involve finding a feasible solution. We assume that the problem has a search space that is bounded by the maximum depth of the search tree.

Searching under a time constraint is different from searching without one. When no time constraint is imposed, one is interested to find the optimal solution in the memory space allowed. On the other hand, when a deadline is included, one is interested in designing an algorithm that can find the best solution in the time limit. In our previous work, we have developed Real-Time Search [1], a class of algorithms for finding good solutions when feasible solutions are easy to find. We have shown that the solution found is very close to the best possible solution that can be found in the time limit by any other practical search algorithm.

In this paper, we are interested in designing a search algorithm for solving constrained optimization problems in such a way 1) that this algorithm will have a good chance of finding a feasible solution if a solution can be found in the time limit by any practical search algorithm, and 2) that this algorithm will require the minimum additional amount of time to find a feasible solution if a solution cannot be found in the time limit.

We focus our discussion on tree searches, although our algorithm can also be applied to graph searches with minor modifications. A search tree in this paper can be considered as a collection of search nodes related by ancestor-descendent relationships. Determining the traversal order of a search tree, however, is different from scheduling a collection of tasks with precedence relationships. In our case, nodes in a search tree are not generated until they are explicitly expanded, and expanding nodes in the search tree amounts to solving the application problem. As a result, conventional scheduling algorithms that rely on a priori information cannot be applied. On the other hand, searching under time and space constraints is closely related to solving problems in imprecise computations: it involves a trade-off between the amount of resources allocated and the quality of the solution obtained.

Research on real-time approximate searches for artificial intelligence applications can be categorized in two classes: one class of algorithms that aim at finding heuristic solutions without establishing bounds on solution quality [4, 6-8], and one class that find heuristic solutions with the minimum estimated deviation from the optimal solution [1, 5, 9, 10]. We are interested in the second class, specifically, the solution of constrained optimization problems in limited time. Related and applicable search algorithms include the guided depth-first search (GDFS), best-first search (BFS), and band search [2].

Research supported by National Aeronautics and Space Administration under contract NAG 1-613.

Proc. IEEE Workshop on Imprecise Computation, Pheonix, AZ, Dec. 1992

The objective of solving constrained minimization problems in limited time is not always well defined. It is well defined if there exists a practical search algorithm for finding a feasible solution in the time limit. In this case, we would like to develop a search algorithm that can find a better solution within the time limit. To ensure that we have a common basis for comparison, we assume in this paper that a GDFS guided by lower-bound values of search nodes is used as a reference algorithm. On the other hand, the objective is not well defined when feasible solutions cannot be found in the time limit by any practical algorithm. Here, we are interested to develop a search algorithm that would require the smallest additional overhead to find a feasible solution.

To summarize, we identify three possible objectives of an algorithm for searching in limited time.

1) If a solution can be found by any practical algorithm in the time limit, the algorithm designed should a) find the first solution as quickly as possible, and b) make the first solution as best as possible.

2) If a solution can be found by any practical algorithm in the time limit, then the algorithm designed should a) find better solutions as quickly as possible, and b) improve the *estimated quality* of solutions as quickly as possible.

3) If a solution cannot be found by any practical algorithm in the time limit, then the algorithm designed should make the best progress in finding the first solution; that is, if the search were continued, our algorithm should require the minimum amount of time (among all practical algorithms) to find the first solution.

It is very difficult, however, to design a algorithm that achieves all these objectives simultaneously. We anticipate that such an algorithm would have two components, one tries to find the first solution, and another tries to refine the first solution once it is found. In this paper, we assume that a GDFS is used to find the first solution, and we propose efficient search algorithms for achieving objective (2). Future studies will address algorithms for achieving the other objectives.

2: Band Search

Band search is an algorithm that generalizes GDFS and BFS [2]. It works on a search tree of D levels, that is, a tree of height D. The search allocates a band of at most W nodes in each level of the search tree for storing active nodes in that level, D priority lists, one for each level, for storing overflow nodes, and D counters, one for each level, for keeping track of and to limit the degree of backtracking allowed in each level. The algorithm has three major features: 1) it selects for expansion in a best-first fashion from all nodes in the bands; 2) it moves nodes from an overflow list into the corresponding band in a depth-first fashion; and 3) it restricts backtracking so that at most W nodes in a level are fully searched before allowing new nodes in this level into the band. Band search specializes to be a GDFS when W is one, and a BFS when W is unlimited. By choosing W appropriately, band search can often out-perform GDFS and has performance very close to that of BFS. The algorithm is static in the sense that the size of each band is kept constant during the search.

In this paper, we extend band search so that the band size can change dynamically during the search; we also consider hybrid search algorithms using GDFS and band search. We describe these algorithms in the following.

1) *Static uniform band search* (sUBS). This is the original version of band search [2] that keeps the size of each band constant throughout the search. Let sUBS(W) be such a search with band width W.

2) *Dynamic band search* (dBS). In a dBS, the band width changes dynamically during the search. Initially, dBS(W) has a band width of one for all levels. During the search, the band width is changed when either feasible or infeasible solutions are found.

   a) *Increasing the band width.* If all descendents of a non-terminal node i in level d are infeasible, then we interpret that the guidance function has not performed well in the band width allowed. To overcome this problem, we increase the band width by one for all levels from 1 to $d-1$. If the band width is larger than a predefined threshold W, then we reset it to the threshold. For each band whose band width is increased, we add nodes from the overflow list into the band.

   b) *Decreasing the band width.* If a feasible solution is found when expanding a non-terminal node, then we interpret that the guidance function has performed well, and we may reduce the band width without degrading its performance. In this case, we decrease the band width by one for all levels from 1 to $d-1$, except in the case where the band width is already at one. Note that for each level whose band width is reduced, we do not move nodes from the band to the overflow list.

3) *Hybrid GDFS-Band Search.* A hybrid search is one that starts with a GDFS and switches to a band search (either sUBS or dBS) later on. The rationale lies in the fact that a GDFS usually performs well in finding the first solution, while a band search is good in refining an existing solution and in finding the optimal solution. We denote these algorithms as GDFS_sUBS(W) and GDFS_dBS(W), respectively.

All the search algorithms described above can find the optimal solution of a constrained minimization problem and prove its optimality when the time constraint is large enough. Further, we can derive the quality (worst-case approximation degree) of an incumbent solution by finding the ratio between the value of the solution and the minimum lower bound of any unexpanded node in the search tree. For static algorithms, the quality of the solution found is monotonic with respect to the time allowed. In contrast, dynamic algorithms make decisions based on non-deterministic runtime information; hence, a larger time constraint may not always result in a better solution.

An iterative version of band search not presented here is one that runs multiple sUBS iteratively, increasing the band width in successive iterations. We found that this algorithm does not perform well, especially when the search tree is deep.

3: Experimental Results

In this section, we show the empirical performance of GDFS_sUBS and GDFS_dBS with respect to that of GDFS in solving *maze* and *asymmetric traveling salesperson* (ATS) problems. These problems have sparse solutions, as there are constraints that renders some possibilities infeasible.

A maze problem is to find the shortest path that connects an entrance to an exit in a maze. A maze is generated by a modified version of the X maze program [3], whose original version generates mazes with only one solution. To increase the solution density, walls are removed systematically by generating a random number in [0,100] and by testing to see whether it is less than a threshold v. We denote maze_v(n,s) as an n-by-n instance of the maze problem with threshold v and random seed s. The lower bound of a node in the search tree of this problem is calculated as its Manhattan distance between the current position and the exit.

An ATS problem is to find the shortest cyclic tour covering n cities, where a city may not be fully connected to all other cities. We assume that the triangular inequality is satisfied when three cities are connected to each other. This property is enforced during the generation of sample problems: all cities are first mapped to a Cartesian plane before their distances are calculated; a link is then removed if a random number in [0,100] is *larger* than a threshold v. We denote the resulting problem as ATS_v(n,s), where v is the threshold, and s is the seed. The lower bound of a node in the search tree is evaluated by finding the cost of the minimum spanning tree.

Since GDFS has been tested empirically to find the first solution quickly, we apply GDFS in both GDFS_sUBS and GDFS_dBS to find the first solution s_1. Let this time be T_1.

With respect to objective (2) stated in Section 1, we show empirical performance results of GDFS_sUBS and GDFS_dBS with respect to GDFS over a wide range of time constraints. The quality of a solution is measured by its *exact approximation degree* which is defined as $s/s^* - 1$, where s is the solution value returned by the method under consideration, and s^* is the value of the optimal solution. Let E_{GDFS}, E_{GDFS_sUBS}, and E_{GDFS_dBS} denote the exact approximation degrees achieved by algorithms GDFS, GDFS_sUBS, and GDFS_dBS, respectively. The cost of an algorithm is measured by the *maximum amount of memory* required to store its active search nodes (time is not part of the cost function as a time constraint is specified ahead of time). Let M_{GDFS}, M_{GDFS_sUBS}, and M_{GDFS_dBS} be the amounts of memory required by GDFS, GDFS_sUBS, and GDFS_dBS, respectively, for storing search nodes.

To compare the performance across different problem instances, we need to normalize time, since T_1 and T_{opt} (time for GDFS to find the optimal solution but not prove its optimality) may vary over several orders of magnitude. One possible way is to normalize time constraint T to $(T-T_1)/(T_{opt}-T_1)$ so that T_1 and T_{opt} correspond to 0 and 1, respectively. We are only interested in the time interval between T_1 and T_{opt}, because a) $T_1 \ll T_{opt}$ (as it is true for 18 out of 20 random instances we have experimented); and b) hybrid band searches use GDFS for finding the first feasible solution; consequently, they all have the same behavior before T_1.

In Figures 1 and 2, we show the solution quality Ξ $(=E_{GDFS}-E_{GDFS\_sUBS(10)})$ and memory space used Π $(=M_{GDFS\_sUBS(10)}/M_{GDFS})$ of GDFS_sUBS(10) with respect to GDFS for various normalized times of 5 random instances of each of the ATS and maze problems. In Figures 3 and 4, we show the quality and cost of GDFS_dBS(10). In these searches, we have chosen the band width so that that a reasonable amount of memory space is needed. Due to space limitation, we do not

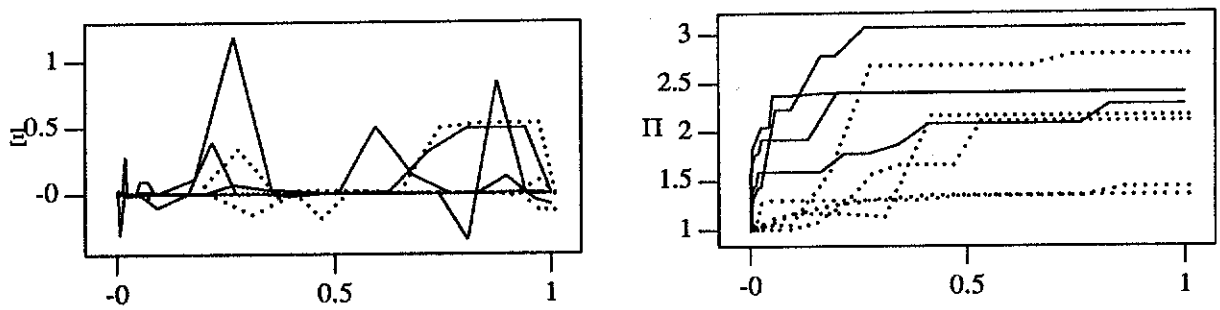

Figure 1. Performance and cost of GDFS_sUBS(10) in solving 5 instances of Maze_20 (dotted curves) and Maze_30 (solid curves), where $\Xi \triangleq E_{GDFS} - E_{GDFS\_sUBS(10)}$ and $\Pi \triangleq M_{GDFS\_sUBS(10)} / M_{GDFS}$.

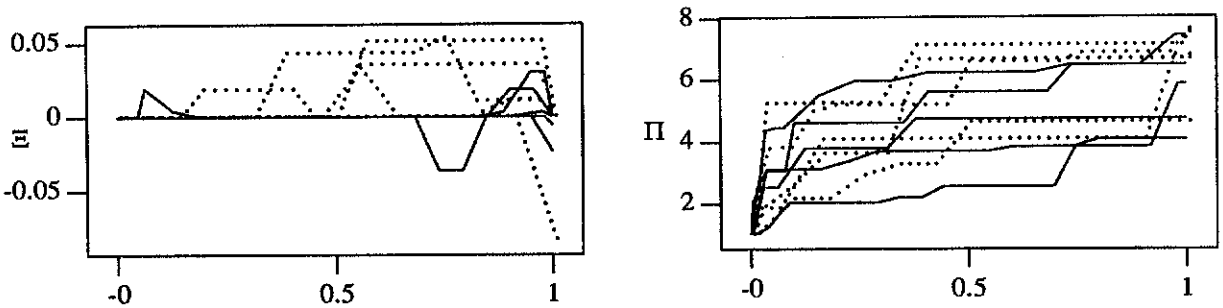

Figure 2. Performance and cost of GDFS_sUBS(10) in solving 5 instances of ATS_10 (dotted curves) and ATS_15 (solid curves), where $\Xi \triangleq E_{GDFS} - E_{GDFS\_sUBS(10)}$ and $\Pi \triangleq M_{GDFS\_sUBS(10)} / M_{GDFS}$.

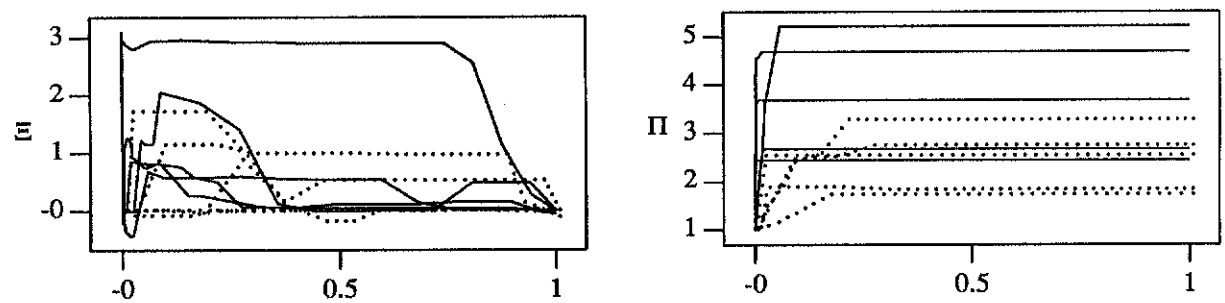

Figure 3. Performance and cost of GDFS_dBS(10) in solving 5 instances of Maze_20 (dotted curves) and Maze_30 (solid curves), where $\Xi \triangleq E_{GDFS} - E_{GDFS\_dBS(10)}$ and $\Pi \triangleq M_{GDFS\_dBS(10)} / M_{GDFS}$.

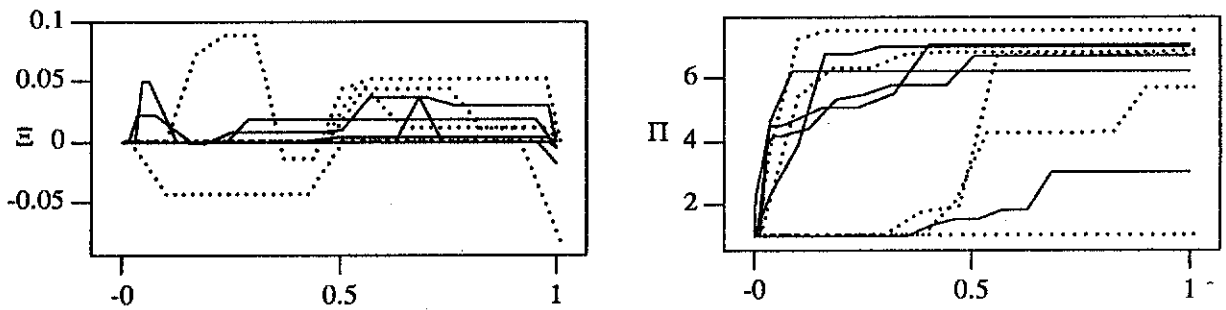

Figure 4. Performance and cost of GDFS_dBS(10) solving 5 instances of ATS_10 (dotted curves) and ATS_15 (solid curves), where $\Xi \triangleq E_{GDFS} - E_{GDFS\_dBS(10)}$ and $\Pi \triangleq M_{GDFS\_dBS(10)} / M_{GDFS}$.

show the performance of our algorithms with different thresholds. The performance is similar to the results shown here.

We observe in these graphs that the quality of the hybrid band search is high when a solution has a lower exact approximation degree. The curves above the x axis correspond to solutions found with quality better than that of GDFS. These graphs also show that the costs incurred by GDFS_sUBS(10) and GDFS_dBS(10) are reasonable. (Theoretically, GDFS_sUBS(10) and GDFS_dBS(10) incur no more than 10 times the maximum memory space incurred by GDFS.)

These figures show that GDFS_sUBS(10) and GDFS_dBS(10) generally perform better than GDFS, as there are more curves above the x axis. Anomalies may sometimes happen and result in quality curves below the x axis [2]; however, they happen less frequently when the band width is increased. Our experimental results further show that GDFS_dBS(10) performs better GDFS_sUBS(10). One possible reason is that the dynamic algorithm can collect run-time information and adjust itself to make better decisions.

Finally, we observe in these graphs that the algorithms proposed are robust. This is illustrated by the fact that the dotted and solid curves, which correspond to the performance of searching with accurate and less accurate guidance functions, are generally above the x axis.

References

[1] L.-C. Chu and B. W. Wah, "Optimization in Real Time," *Proc. Real Time Systems Symp.*, pp. 150-159, IEEE, Nov. 1991.

[2] L.-C. Chu and B. W. Wah, "Band Search: An Efficient Alternative to Guided Depth-First Search," *Proc. Int'l Conf. on Tools for Artificial Intelligence*, IEEE Computer Society, (accepted to appear) Nov. 1992.

[3] R. Hess, D. Lemke, and M. Weiss, "Maze," *X Version 11 Release 4*, July 1991.

[4] R. E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, vol. 42, pp. 189-211, Elsevier Science Publishers, 1990.

[5] E. L. Lawler and D. W. Wood, "Branch and Bound Methods: A Survey," *Operations Research*, vol. 14, pp. 699-719, ORSA, 1966.

[6] V. R. Lesser, J. Pavlin, and E. Durfee, "Approximate Processing in Real-Time Problem Solving," *AI Magazine*, pp. 49-61, AAAI, Spring 1988.

[7] C. J. Paul, A. Acharya, B. Black, and J. K. Strosnider, "Reducing Problem-Solving Variance to Improve Predictability," *Communications of the ACM*, vol. 34, no. 8, pp. 80-93, Aug. 1991.

[8] S. Shekhar and S. Dutta, "Minimizing Response Times in Real Time Planning and Search," *Proc. of AAAI-89*, pp. 238-242, 1989.

[9] B. W. Wah and L.-C. Chu, "TCA*--A Time-Constrained Approximate A* Search Algorithm," *Proc. Int'l Workshop on Tools for Artificial Intelligence*, pp. 314-320, IEEE, Nov. 1990.

[10] B. W. Wah and L.-C. Chu, "TCGD: A Time-Constrained Approximate Guided Depth-First Search Algorithm," *Proc. Int'l Computer Symp.*, pp. 507-516, Tsing Hua Univ., Hsinchu, Taiwan, Dec. 1990.