N88-16410

ITERATIVE-DEEPENING HEURISTIC SEARCH FOR OPTIMAL AND SEMI-OPTIMAL RESOURCE ALLOCATION

Susan M. Bridges
James D. Johannnes
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

This paper examines the use of iterative-deepening A* (IDA*) in solving a certain class of resource allocation problems. IDA* stores a number of nodes proportional to the depth of the solution in the search tree rather than the number of nodes generated as in A*. A* and IDA* solutions to the resource allocation problem were implemented with empirical results showing that the low effective branching factor associated with real-valued cost estimates causes an unacceptably large number of nodes to be generated by IDA*. A modification of IDA* that can be used for semi-optimization, called IDA*$_\varepsilon$, was developed and is shown to be an ε-admissible tree search algorithm. A comparison of the performance of A*, IDA*, and IDA*$_\varepsilon$ for resource allocation demonstrates the effectiveness of IDA*$_\varepsilon$ in reducing the computational requirements of the problem.

1. INTRODUCTION

Mission planning for space applications must address many resource allocation problems of various types. This paper will examine a class of resource allocation problems in which resources have a given effectiveness in reducing the value of tasks and more than one resource may be used for a particular task. The problem is to find an optimal allocation of resources to tasks that maximizes gain, i.e. the total reduction in task value minus the cost of the resources allocated.

Slagle and Hamburger [6] report the use of the A* heuristic search algorithm in solving this resource allocation problem. In recent work by Korf [1,2], a variation on A*, called iterative-deepening A* (abbreviated IDA*) has been shown to be optimal in terms of space and time requirements among heuristic best-first tree searches. This paper reports the investigation of IDA* for solving this type of problem. Difficulties that arise when the cost function used with IDA* is real-valued rather than integer are discussed, and the development of modifications of IDA* that can be used for semi-optimization is reported.

2. BACKGROUND

The following description of this type of non-linear resource allocation problem is modeled after that of Slagle and Hamburger [6]. The problem is to assign a set of resources to a set of tasks where each resouce has a given cost and each task a given value. Additionally, for each resource-task pair, an effectiveness value in the range 0.0

to 1.0 is given that represents the expected portion by which the task value will be reduced if the resource is assigned to the task. For each resource, there are n + 1 choices for assignment where n is the number of tasks, i.e. the resource may be assigned to any of the n tasks or not used at all. The choice of not using a resource is assumed to have a gain of 0 and so would be chosen over assignments resulting in a negative gain. Given a choice between not using a particular resource or using a resource-task assignment with a gain of 0, the choice of not using the resource will be chosen. Several resources can be assigned to the same task with the assumption that they do not interact.

3. A* VERSUS IDA*

Slagle and Hamburger [6] describe the use of the A* algorithm to find an optimal plan for assignment of resources to tasks. A* has been shown to always yield an optimal solution when the heuristic cost function used is consistently "optimistic" [3], i.e. for minimization problems it always underestimates the cost of the solution and for maximization problems it always overestimates the value of the solution. Heuristics with this property are called "admissible" . In general, the A* algorithm has storage requirements proportional to the number of nodes that will be expanded. Korf [1,2] has shown that a modification of A*, depth-first iterative-deepening A* (IDA*), has space requirements proportional to the depth of the solution node in the search tree. In addition, IDA* always finds an optimal solution in a manner similar to A* and asymtotically expands the same number of nodes as A*. In the section below, we describe the implementations and results of experiments comparing the efficiency of the A* and IDA* algorithms for resource allocation.

For both algorithms, the search tree is organized such that each level in the tree represents the allocation of a particular resource. Thus, the depth of the tree, d, is equal to the number of resources, and the branching factor of the tree is (n + 1) giving a total of $(n + 1)^d$ nodes in the complete search tree. Each node in the tree represents a partial plan for allocation of resources with the root node representing the plan of not using any of the resources. The evaluation function for each node is the sum of the gain (g) achieved by the partial plan, and an estimate of the gain achievable by allocation of the remaining resouces (h). The heuristic used for the calculation of h in this implementation is similar to one of the heuristics described by Slagle and Hamburger [6]. For each resource remaining to be allocated, the maximum gain achievable by that resource with the task values of the current node is calculated. Interaction among resources is ignored. The value of h is the sum of these maximum gains or the sum of the current task values, whichever is lower. Clearly, the cost function is both monotone and admissible since the gain calculated is always optimistic and becomes more accurate (lower) as more resources are added to the partial plan.

In the A* solution to this problem, a priority queue (OPEN list) is maintained that contains all nodes that have been generated but not expanded. These nodes are ordered in descending order by estimated gain. Nodes are successively removed from the queue and expanded until a solution node is found (a node where the actual gain is equal to the estimated gain). This problem is unusual in that every node in the search

tree represents a solution (an allocation of resources to tasks), so any child node that has an estimated gain less than an actual gain already found can be pruned and never reconsidered.

The IDA* algorithm does not use an OPEN list. Instead, a threshhold value is maintained for each iteration and any node that has an estimated gain less than the threshhold value is not generated on that iteration. The initial value for the threshhold is the gain estimate for the root node. A simple recursive depth-first search is done of all nodes with gain estimates greater than or equal to the cutoff. In addition, the value of the highest rejected node is recorded. The search terminates when a node is found with an actual gain equal to its estimated gain. If a solution has not been found at the conclusion of the depth-first search with a given cutoff, the value of the threshhold is changed to the highest rejected value and the search is done again. The iteration process is repeated until a solution is found and the solution found is guaranteed to be optimal since all nodes with higher possible gains will have been examined on previous iterations.

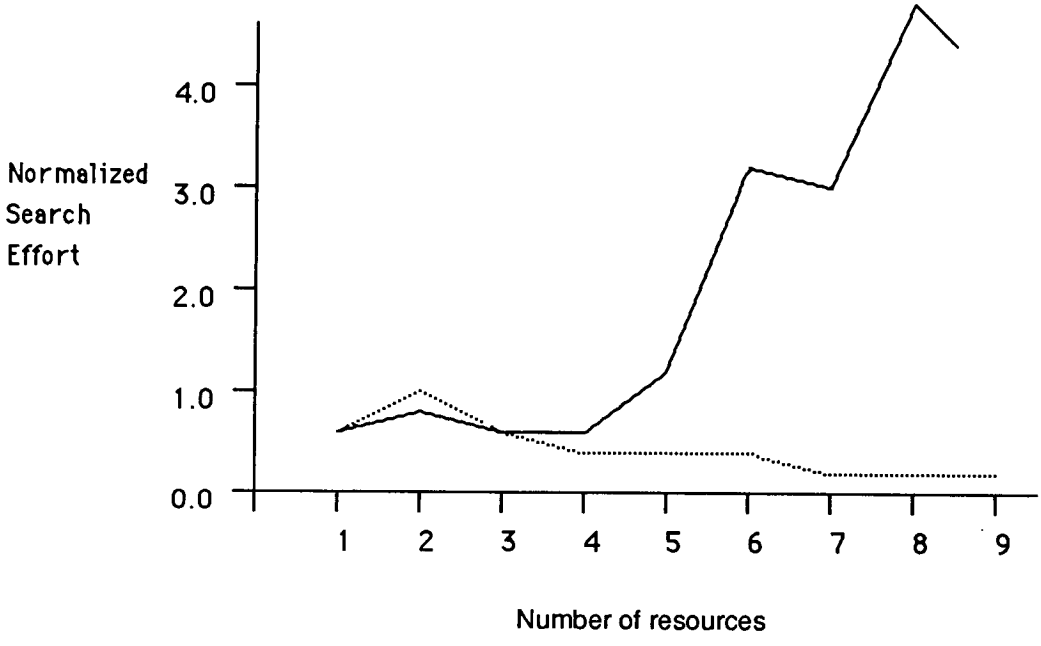

Figure 1. Performance comparison of A*, IDA*, and IDA*$_\varepsilon$ on the same data sets. In this experiment, the number of resources was equal to the number of tasks. Task values, resource costs, and effectiveness values were randomly chosen from uniform distributions. The solid line represents the mean ratio of the number of nodes expanded using IDA* compared to A*. The dotted line represents the mean ratio of the number of nodes expanded using IDA*$_\varepsilon$ compared to A*.

The solid curve in Figure 1 shows the results of an experiment comparing the use of A* and IDA* on the same randomly generated data sets. The curve represents the mean ratio of the number of nodes expanded using IDA* to that expanded using A*. Although IDA* always stores far fewer nodes than A*, the number of nodes generated by IDA* grows at a much faster rate than for A*. Both algorithms quickly overwhelm

computational resources with A* consuming all available space and IDA* requiring an unreasonable amount of time for relatively small size problems. Other experiments have shown that some improvement in performance is achieved by considering the resources in order of maximum possible gain, but the improvement is not sufficient to avoid the computational limits encountered.

Korf [2] has shown that for tree search problems with a branching factor, b, of magnitude greater than 1, IDA* opens asymtotically the same number of nodes as A*. The order of the algorithm is b^d, where d is the number of iterations and b is the ratio of the number of nodes opened during the current iteration to the the number of nodes opened on the previous iteration. The constant coeffficient is $(1 - 1/b)^{-2}$ as the search depth goes to infinity. Korf [1] pointed out that for branching factors close to 1, the constant coefficient approaches infinity as the search depth goes to infinity. Unfortunately, in our problem and, in fact, in many domains with real-valued evaluation functions, the branching factor is often close to 1 with the algorithm opening only one additional node on each iteration. The semi-optimization algorithm described in the next section was developed in an attempt to find a modification of IDA* that has better performance characteristics for problems with real-valued cost functions.

4. SEMI-OPTIMIZATION WITH IDA*

Pearl [4] describes several speedup versions of A* (called A^*_ε) that can be used for semi-optimization. Algorithms that guarantee that the cost of the solution (for minimization problems) will not exceed the cost of an optimal solution by a factor of more than $1+\varepsilon$ are called ε-admissible. We describe an ε-admissible modification of IDA* (IDA^*_ε) which exhibits the same characteristics of increased performance as ε-admissible versions of A* but with much reduced storage requirements. For purposes of this discussion, we will describe the algorithm as used for minimization of cost problems as is traditional for A*.

IDA^*_ε works much like IDA* except that the threshhold for the initial iteration is set to $1+\varepsilon$ times the cost of the root node and on successive iterations it is set to $1+\varepsilon$ times the cost of the lowest rejected node from the previous iteration. As with IDA*, on each iteration a simple recursive depth-first search is done of all nodes with cost estimates less than or equal to the threshhold. Note that the storage requirements for IDA^*_ε are proportional to the depth of the solution in the search tree and are handled automatically via the runtime stack. The search terminates when a goal node is chosen for expansion. A relatively straight-forward modification of Pearl's proof for the ε-admissibility of A* can be used to prove the ε-admissibility of IDA^*_ε.

The only modification necessary to convert the IDA* resource allocation program to

IDA^*_ε was to change the calculation of the threshhold. As guaranteed by the algorithm, the IDA^*_ε program found allocation plans with gains within 10%(ε) of the optimal gains for plans found by A* and IDA*. Since the number of iterations of IDA^*_ε can never exceed $1/\varepsilon$, this algorithm avoids the explosive growth in the number of nodes generated that occurs with IDA*. Figure 1 shows a comparison of the mean number of nodes generated by IDA* and IDA^*_ε with both normalized to A*. IDA^*_ε also opens significantly fewer nodes than A*. This indicates that A* and IDA* spend a great deal of time discriminating among solutions of approximately the same value. Whereas A* has storage requirements proportional to the number of nodes generated, IDA* and IDA^*_ε both have storage requirements proportional to the depth of the search tree (the number of resources in this case). Thus, IDA^*_ε uses far less time than IDA* and far less space than A* while guaranteeing a solution with a cost within a factor of $1 + \varepsilon$ of the optimal solutions found by these two algorithms. This reduction in time and space complexity allowed the IDA^*_ε program to find semi-optimal solutions to problems that could not be solved using A* or IDA* within the time and space limits imposed.

5. CONCLUSIONS

Resource allocation problems of many types will continue to be of vital importance in mission planning. We have demonstrated that when IDA* is applied to one type of resource allocation problem, it uses far less storage than A* but opens far more nodes and thus has an unacceptable time complexity. This is shown to be due, at least in part, to the low-valued effective branching factor that is a characteristic of problems with real-valued cost functions. The semi-optimal, ε-admissible IDA^*_ε search algorithm that we described was shown to open fewer nodes than both A* and IDA* with storage complexity proportional to the depth of the search tree.

6. REFERENCES

1. Korf, R. E. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence,* 27, 1985, pp. 97-109.
2. Korf, R. E. Iterative-deepening-A*: an optimal admissible tree search. *Proceedings Ninth International Joint Conference on Artificial Intelligence,* Los Angeles, CA, 1985, pp.1034-1035.
3. Nilsson, N. J. *Problem solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.
4. Pearl, J. *Heuristics,* Addison Wesley, Reading Mass. 1985.
5. Pearl, J. and J. H. Kim. Studies in semi-admissible heuristics, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 4, 1982 , pp. 392-399.
6. Slagle, J. R., and H. Hamburger. An expert system for a resource allocation problem. *Communications of the ACM,* 28, 1985, pp. 994-1004.