Experiments in Parallel Heuristic Search
(Extended Abstract)

Lélio de P. Sá Freitas
Valmir C. Barbosa

Programa de Engenharia de Sistemas e Computação, COPPE
Universidade Federal do Rio de Janeiro
Caixa Postal 68511
21945 Rio de Janeiro - RJ, Brazil

Summary

We introduce four parallel variants of the $A^*$ and $BS^*$ heuristic search algorithms, two based on $A^*$ and two based on $BS^*$. In $BS^*$ the search is bidirectional, thus hinting at a possibly higher amenability to parallel processing than in the case of $A^*$. Two of the variants we introduce employ a centralized-list approach, and the other two employ a distributed-list approach. Experiments with randomly generated instances of the Route Planner Problem on an right-processor T-800 Transputer hypercube have indicated a generally superior performance of the centralized-list approaches, and between these a very clear superiority of the variant based on the $A^*$ search.

Introduction

Consider the problem whose solution space is the directed graph $D = (N, E)$, where each member of $N$ stands for a partial solution, or, equivalently, a sub-problem, and the edges in $E$ are directed in order to indicate increasing progress in the description of a complete solution, or, equivalently, decreasing complexity of the sub-problems. A node $n \in N$ may have predecessors $(n' \in N$ such that $(n' \rightarrow n) \in E$) and successors $(n' \in N$ such that $(n \rightarrow n') \in E)$.

To solve the problem whose solution space is $D$ is to find a directed path from the starting node $s$ to one of the nodes in the set of goal nodes $G$, in such a way as to satisfy a certain optimization criterion, usually the minimization of this $s$-to-$G$ path's "cost." Often $D$ is a very large graph, and is not entirely available when the search for an optimal $s$-to-$G$ path is started. Rather, its nodes are generated as needed.

We will in this paper restrict ourselves to the cases in which the "cost" of a directed path is given by the sum of the individual "costs" of its directed edges. With each edge $(n \rightarrow n') \in E$ we then associate a cost $c(n, n')$, and with each node $n \in N$ we associate a cost $f(n, m)$, which is the sum of $c(n, n')$ for all edges $(n \rightarrow n')$ on the optimal $s$-to-$G$ path that includes $m$. Of course the value of $f^*$ is the same for all nodes on a same optimal $s$-to-$G$ path.

A very successful "best-first" strategy to determine an $s$-to-$G$ path that minimizes the value of $f^*$ is the $A^*$ algorithm [1,2], which maintains a tree rooted at $s$. This tree grows outward from $s$ by the generation of all successors of a selected leaf. Whenever a node $n \in N$ is generated a pointer $p(n)$ is set to the predecessor of $n$ from which it was generated.

When during the $A^*$ search a leaf is selected which is found to be one of the nodes of $G$, the search terminates and that node is proclaimed the optimal solution to the problem. Depending on the criterion for selecting leaves for the generation of successors, that node in $G$ is indeed optimal in the sense that an $s$-to-$G$ path where $f^*$ is minimum has been found (and is, in fact, given by an $s$-to-$G$ path in the tree that the search maintained).

When $G$ is a singleton whose only member, say $t$, is known beforehand, two concurrent $A^*$ searches may be advantageous, one starting at $s$ and one at $t$, thereby concurrently generating two trees that somehow meet "halfway" between $s$ and $t$ in the solution space. The potential gain with this dual approach comes from the realization that if a tree of height $h$ has a number of nodes which is $O(H)$, then a tree of height $h/2$ has a number of nodes which is $O(\sqrt{H})$, so at least in principle many fewer nodes are generated with this bidirectional search.

The most successful bidirectional search procedure in the literature (among those which do not employ the so-called "wave-shaping" techniques) is the $BS^*$ algorithm [3], which is a sequential search algorithm, but whose potential for parallelism is obvious. $BS^*$ is essentially composed of two instances of the $A^*$ search, one that generates a tree from $s$ toward $t$, and another generating a tree from $t$ toward $s$. In this paper we describe an experimental evaluation of two parallel versions of $A^*$ and two parallel versions of $BS^*$ on the Route Planner Problem (RPP, to be described later), using a T-800 Transputer hypercube [4] programmed in Occam2 [5].

Although $BS^*$ has in principle a much greater potential for parallelism than $A^*$ does, it is not clear at all which one will perform better in a real distributed parallel imple-

This work was supported by the Brazilian agencies FINEP, CAPES, and CNPq.
The main reason why one should not trust a hasty a priori judgement concerning the two methods' relative performance is that BS*, although capable of generating many fewer nodes due to the existence of the two search trees, has to somehow control the parallel expansion of the two trees so that they are led to meet in the "middle" of the solution space, thereby imposing an additional cost which may affect performance.

Both the A* and the BS* search methods maintain their trees' leaf and nonleaf nodes in lists, which apparently constitutes an inherent "sequential bottleneck" of the methods. As in previous parallel implementations of branch-and-bound techniques [6 9], where similar lists are also kept, a choice has to be made concerning the use of a centralized- or a distributed-list approach. These are then the four parallel heuristic search methods we evaluate: A* with centralized list and distributed list, and BS* with centralized lists and distributed lists. As we shall see, an appropriate partitioning of the node set N has led to our conclusion that for RFP the A* method with distributed list is by far the best of the four approaches.

The paper has for lack of space been considerably shortened; a complete version with details of the sequential and parallel algorithms, as well as a more thorough experimental evaluation, can be found in [10].

In what follows, the tree nodes whose successors have already been generated are called closed nodes; the others are referred to as open nodes.

The Parallel Versions of A*

Our parallel algorithms employ a set of \( \tau \) concurrent tasks \( T = \{t_1, \ldots, t_{\tau}\} \), each of which responsible for a subset \( N_i \subseteq N \) of nodes in the solution space of the problem, \( 1 \leq i \leq \tau \). These subsets \( N_i \) exhaust the space \( N \) and are usually disjoint, although the latter is not a requirement. The idea behind this partition of \( N \) into \( \tau \) sets is that task \( t_i \) will only generate nodes in \( N_i \).

Parallel A* with Centralized List (PACL)

In a centralized-list, parallel implementation of A* an additional concurrent task, \( t_M \), the master task, is utilized. What this task does is to select an open node and send it to each of the tasks in \( T \) for successor generation. Those tasks in turn send back to \( t_M \) the successors generated with their corresponding attributes. After the distributed parallel A* search is terminated, \( t_M \) may then send the solution node \( n \in G \) it found (if any) to any \( t_i \in T \) for retrieval of the entire optimal \( s \)-to-\( G \) path. PACL can be implemented on a hypercube with \( 2^k \) processors by letting the number \( \tau \) of tasks in \( T \) be \( \tau = 2^k - 1 \). The master task \( t_M \) is then allocated to one of the processors, and each of the \( t_i \in T \) is allocated to one of the remaining processors.

Parallel A* with Distributed List (PADL)

In contrast with PACL, in a distributed-list, parallel implementation of A* only the \( \tau \) identical tasks in \( T \) are used, and employ a distributed extreme-finding procedure to determine the open node to have its successors generated.

When the algorithm terminates at all tasks \( t_i \in T \), the optimal \( s \)-to-\( G \) path it found (if any) is readily available at all tasks. On a hypercube with \( 2^k \) processors an implementation of PADL employs \( \tau = 2^k \) tasks, each one running on each of the processors.

The Parallel Versions of BS*

For the parallel versions of BS* two sets of concurrent tasks will be employed, each with \( \tau \) tasks. These are \( T_s = \{t_{s,1}, \ldots, t_{s,\tau}\} \) for the search that starts at \( s \) and \( T_t = \{t_{t,1}, \ldots, t_{t,\tau}\} \), for the search that progresses in the opposite direction from \( t \). As in the case of A*, a subset \( N_i \) of \( N \) is associated with \( t_{s,i} \) and \( t_{t,i} \), which will then both generate nodes in \( N_i \) only.

A task \( t_{s,i} \in T_s \) will often have to communicate with its counterpart \( t_{t,i} \in T_t \). In our parallel algorithms this will happen whenever a new list of successors is generated by \( t_{s,i} \) and \( t_{t,i} \), so the two trees will be expanded in a relatively synchronized fashion, which is also convenient from the standpoint of the intuitive expectation that for efficiency the two trees should meet approximately "halfway" between \( s \) and \( t \).

Parallel BS* with Centralized Lists (PBSCCL)

For a parallel version of BS* with centralized lists two master tasks, \( t_{s,M} \) and \( t_{t,M} \), are used. As in the case of A*, these tasks interact with those in \( T_s \) and \( T_t \) for node generation. When the algorithm terminates, the optimal \( s \)-to-\( t \) path it found (if any) can be traced back from the meeting node of the two tree expansions by any pair \( t_{s,i} \) and \( t_{t,i} \) of tasks. In a \( 2^k \)-processor implementation of PBSCCL we let \( \tau = 2^k - 1 \). The tasks can then be allocated as follows. \( t_{s,M} \) runs on processor 0 and \( t_{t,M} \) on processor \( 2^k - 1 \). Allocate each of the \( t_{s,i} \) (in increasing order of \( i \)) to each of the processors \( 1, \ldots, 2^k - 1 \), and each of the \( t_{t,i} \) (in increasing order of \( i \)) to each of the processors \( 2^k - 1 + 1, \ldots, 2^k - 1 \). Then for any \( i \) there exists a communication channel connecting the processor on which \( t_{s,i} \) runs to that where \( t_{t,i} \) runs.

Parallel BS* with Distributed Lists (PBSDL)

As in the case of the parallel algorithm for A* with distributed lists, here too the tasks in \( T_s \) and \( T_t \) execute a distributed procedure to determine the open node to have its successors generated on the trees rooted at \( s \) and \( t \), respectively. Like PBSCCL, here too pairs of tasks \( t_{s,i} \) and \( t_{t,i} \) communicate. When the algorithm terminates, the optimal \( s \)-to-\( t \) path it found (if any) can be traced back from the meeting node of the two tree expansions by any pair \( t_{s,i} \) and \( t_{t,i} \) of tasks, as in the case of PBSCCL. A hypercube implementation with \( 2^k \) processors employs \( \tau = 2^k - 1 \) tasks for each tree expansion. Each task \( t_{s,i} \) (in increasing order of \( i \)) can be allocated to each of the processors \( 0, \ldots, 2^k - 1 \), and each \( t_{t,i} \) (in increasing order of \( i \)) to each of the processors \( 2^k - 1, \ldots, 2^k - 1 \). As in the case of PBSCCL, a communication channel exists between the processor at which \( t_{s,i} \) runs and that at which \( t_{t,i} \) runs.
Experimental Evaluation on RPP

Given a set of points \( Z \) and two distinguished points \( z, z' \) from \( Z \), RPP asks for the shortest route between \( z \) and \( z' \) with respect to a set of positive distances \( \text{dist}(a, b) \) between each two points \( a, b \in Z \).

The set of distances \( \text{dist}(a, b) \) implies an undirected graph whose vertices are the elements of \( Z \) and whose edges correspond to those pairs of points \( a, b \in Z \) such that \( \text{dist}(a, b) < \infty \). In terms of our heuristic search terminology, this undirected graph is the underlying structure of the solution space \( D \), where \( N = Z \) and \( E = \{(a \to b), (b \to a) \mid \text{dist}(a, b) < \infty \} \).

In this paper we take the points in \( Z \) to be points on a plane, and the distance \( \text{dist}(a, b) \) to be the Euclidean distance between \( a \) and \( b \) if \( \text{dist}(a, b) < \infty \). In terms of our previous notation, we then have \( s = z, t = z' \), \( c(n, n') = \text{dist}(n, n') \) for all \( (n \to n') \in E \), and the function \( f^* \) shortest distances.

The experiments we report on here have been carried out on an eight-processor, T-800 Transputer hypercube, with the four parallel algorithms programmed in Oceana2. Our experiments have been performed on random undirected graphs with points in \( Z \) generated randomly inside a \( 100 \times 100 \) square, and pairs of points (independently) interconnected with probability \( P_e \). We report on experiments for \( |Z| \in \{50, 60, 70, 80, 90, 100\} \), \( P_e \in \{0.1, 0.3, 0.5\} \), and \( 2^k = 8 \).

Once the number of tasks has been determined, the specification of each \( N_c \) constitutes another crucial step toward a real implementation. As each \( N_c \) contains the set of nodes to be generated by a task, its determination is in fact a load balancing procedure that should attempt to get as close as possible to the ideal situation in which all tasks are required to do approximately the same amount of node-generation work.

For our experiments on RPP we have chosen to proceed as follows. As the points in \( Z \) are randomly generated, they are assigned circularly to the tasks (the \( N_c \)'s), i.e., the first point is assigned to the first task, the second point to the second task, and so on, until all tasks have one point, then the first task receives another point, etc. Points are then interconnected pairwise with probability \( P_e \), so the average number of points interconnected to a point \( a \in Z \) and allocated to a certain task \( t \) is the same for all tasks. This means that the successors of a node \( n \in N \) are on average distributed evenly among the tasks, whose processors are then on average loaded equally.

Each run of an algorithm in our experiments involves taking \( s \) and \( t \) to be all the possible pairs of nodes, thereby avoiding a possible biased choice for \( s \) and \( t \) in the randomly generated set of points. Partial results are shown in Figures 1(a) through 1(e), where the speedup of the algorithms with respect to the sequential \( A^* \) search is shown on eight processors organized as a hypercube.

A general trend in the performance of the four parallel methods is that it tends to increase with the problem's density (i.e., the value of \( P_e \)) and with problem size. This is in a way expected, since as either \( P_e \) or problem's size increases the methods become more "computation-intensive" with the generation of more nodes, and consequently the speedup degradation caused by interprocessor communication is less effective.

Another general trend that can be inferred is the superiority of the distributed-list approaches over their centralized-list counterparts. This too is expected, in view of the obvious communication bottlenecks with the master tasks in the centralized-list approaches. An exception to this general observation has been the case of instances with \( P_e = 0.1 \) on eight processors (Figure 1(a)), where PBSDL exhibits a performance slightly better than PBSCL. This may be an indication (confirmed by the results for
other network sizes) that for low densities the communication between tasks operating on different "wave fronts" of BS* is the predominant source of communication overhead, so the centralized- and distributed-list approaches tend to perform equivalently to each other.

If we restrict ourselves to observing the centralized list methods, then clearly the BS* implementation is superior to that of A*. We believe the reason for this to be twofold. First of all, each tree expansion in PBSCL spans half of the processors used by PACL, so the communication bottleneck with the master task is much less prominent in the former case. Secondly, we might also think of attributing a fraction of this superiority of PBSCL over PACL to the very nature of the BS* search, with the expected superior performance of two concurrent tree expansions. However, if now we restrict ourselves to observing the performance of the distributed-list methods alone, then we are led to conclude that perhaps this second reason is in fact of minor importance after all, as PADL is in general far superior to all others.

Concluding Remarks
We have in this paper investigated four parallel algorithms for A* and BS* heuristic searches. Two of these algorithms follow a centralized-list approach (PACL for A*, PBSCL for BS*), and two employ distributed lists in conjunction with a distributed extreme-finding procedure (PADL for A*, PBSDL for BS*).

Experiments have been described for instances of the Route Planner Problem (RPP) on an eight-processor, T-800 Transputer hypercube. These experiments have indicated a tendency for increased performance of all four algorithms as either the problem's density or problem size increases. They also reveal a general superiority of the distributed-list approaches over the centralized-list approaches, although apparently this tends to be the other way around for BS* when dealing with low-density instances of RPP. Within the framework of the centralized-list methods, BS* seems to be superior to A*. When restricted to the distributed-list approaches, however, A* is considerably superior to BS*.

Although the performance of heuristic search methods is very dependent upon the particular problem being treated and the particular instances of that problem as well, and although (potentially serious) limitations have been imposed by our hardware's relatively small number of processors and memory size per processor, the overall superiority of PADL seems to be well established.

References