Proceedings of the Third Biennial Conference of the Canadian Society for Computational Studies of Intelligence, Saskatoon, 14-16 May 1980, pp. 232-239.

APPLICATIONS OF THE CONTRACT NET FRAMEWORK: SEARCH

Reid G. Smith

Defence Research Establishment Atlantic
Box 1012
Dartmouth, Nova Scotia, B2Y 327, Canada.

Abstract[1]

We discuss the implementation of heuristic search algorithms in a distributed problem solver whose processors interact according to the *contract net* protocol. Task distribution is viewed as a local mutual selection process based on a two-way transfer of information between processors with tasks to be executed and processors with knowledge-sources capable of executing those tasks.

As an example of the approach, we consider the N Queens problem. We then derive measures of the speedup that can be expected from the application of a distributed processor to search problems that involve regular trees, and discuss the effect of coupling between processors on speedup. Bounds are developed for the number of processors that are required to achieve maximum speedup.

1 Introduction

Distributed problem solving is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge-sources (KSs), each of which may reside in a distinct processor node. The KSs *cooperate* by sharing tasks and/or results. By *decentralized* we mean that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. *Loosely coupled* means that individual KSs spend most of their time in computation rather than communication. Such problem solvers offer the promise of speed, reliability, extensibility, the ability to handle applications with a natural spatial distribution, and the ability to tolerate uncertain data and knowledge.

Search problems are attractive as applications of distributed problem solving for three major reasons. First, exploration of a search space of the size commonly encountered in AI applications consumes a large amount of computing time (see, for example, discussions of **Meta-Dendral [Buchanan,** 1978], and **CONGEN [Carhart, 1976]**). Thus, the speedup promised by the distributed approach is attractive. Second, search problems are often modular in form. Numerous relatively independent subtasks are created during the course of a search. These subtasks are ideal candidates for distribution to individual processors. Finally, search is one of the major problem-solving paradigms. It is therefore important to develop tools for applying the new VLSI technology to search problems.

2 Task-Sharing And Contract Negotiation

The *contract net* protocol **[Smith, 1978]**, **[Smith, 1979]** facilitates cooperation of multiple processors in the solution of a problem. Dynamic matching of tasks and KSs is effected by negotiation. A contract is an explicit agreement between a processor that generates a task (the manager) and a processor willing to execute the task (the contractor). (Note that a processor is assumed to contain one or more KSs.) A contract is normally established by a process of local mutual selection based on a two-way transfer of information. In brief, available contractors evaluate task announcements made by several managers until they find one of interest. They submit a bid for that task. The manager then evaluates the bids received from potential contractors and selects the one it determines to be most appropriate. Both parties to the agreement have evaluated the information supplied by the other and a mutual selection has been made. Control is distributed because processing and communication are not focussed at particular processors, but rather every processor is capable of accepting and assigning tasks.

Contract net messages contain slots for information that aids negotiation. A task announcement contains three such slots. The **eligibility specification** is a list of criteria that a processor must meet to be eligible to submit a bid. It enables a processor receiving the message to decide whether or not it is able to execute the task. This specification reduces message traffic by pruning processors whose bids would be clearly unacceptable. The **task abstraction** is a brief description of the task to be executed. It enables a processor to rank the announced task relative to other announced tasks. An abstraction is used rather than a complete description in order to reduce the length of the message. The **bid specification** is a description of the expected form of a bid. It enables a processor to include in a bid only the information about its

[1] This work was supported in part by the Advanced Research Projects Agency under contract MDA 903-77-C-0322, and the National Science Foundation under contract MCS 77-02712. Some of the work described is being pursued in collaboration with Randall Davis at MIT. Joe Maksym also made a number of valuable comments.

capabilities that is relevant to the task rather than a complete description (called a **node abstraction**). This simplifies the task of the manager in evaluating bids and further reduces message traffic.

3 Distributed Search: Overview

In this section we discuss some general characteristics of distributed search. We then show how the contract net protocol can be used to organize a distributed problem solver to perform such a search.

Consider the exhaustive search of a tree in a distributed processor. To make clear the flow of the search, we make the following assumptions: (i) the basic task for each processor is generation of a successor node in the tree, (ii) as soon as a processor generates a node, it distributes that node to another processor for further expansion, (iii) generation of each node requires a constant processing time, (iv) there is a sufficient number of processors so that the expansion of a node can be commenced by one processor as soon as the node has been generated by another, (v) a processor can distribute a successor node to another processor concurrently with generation of another successor node, and (vi) distribution of a node to a processor and reporting of results require a negligible amount of time compared to the time required to expand the node.

The flow of the search process is shown in Figure 1 for a regular tree of branching factor 2 and depth 3. The numbers inside each node circle indicate the time unit at which the node was generated and the processor that generated it (in the format "time / processor").

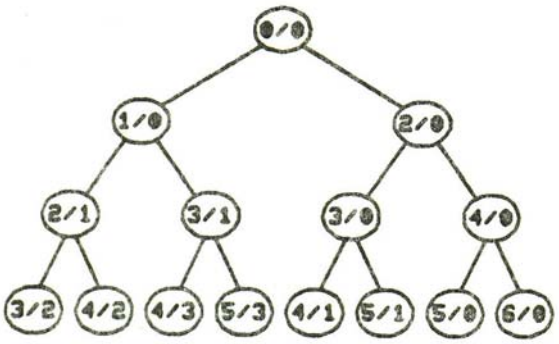

Figure 1. Distributed Search Of A Regular Tree.

At time 1, one successor of the root node is generated. This successor is distributed (we assume instantaneously) to another processor, so that at time 2, two successors are generated. The number of processors involved in the search rises from 1 to 4 and then decreases again to 1 before completion.

It is apparent that problems that entail a large amount of search are especially amenable to a distributed approach – they have the potential for large speedups. In addition, trees comprised of *OR* nodes lend themselves more readily to concurrent exploration than do those comprised of *AND* nodes. This is because less processor synchronization is required for their exploration. Trees with *Ordered-AND* nodes (i.e., nodes that must be expanded in a particular order) are the least amenable to concurrent exploration because they require the greatest amount of synchronization (which inevitably means that some processors will stand idle waiting for results to be generated by other processors).

In Appendix A we present performance measures for exhaustive distributed search of regular trees. It is shown that speedups that are close to linear in the number of processors are possible. It has been shown elsewhere (e.g., **[Imai, 1979]**) that better than linear speedups are possible. This result follows for problems in which application of multiple processors can eliminate fruitless expansion of a large number of nodes. We also see from the analysis that trees that are bushy near the root lead to larger speedups than trees in which bushiness occurs at larger depths. This is due to the fact that more processors get involved quickly, and the nodes they generate can often be queued for later expansion with no increase in search time (because of the decreased demand for processors as the search nears completion). Finally, it is shown that loose-coupling must be maintained if maximum speedups are to be achieved.

3.1 Node Selection

In a distributed search, selection of nodes for expansion and generation of their successors are asynchronous, local processes. Node selection is especially different from the uniprocessor case, where a global evaluation function is used to select one node to be expanded next. Distributed search strategies have a local character because many nodes may be selected concurrently for expansion by individual processors, usually based on a more local evaluation.

If interprocessor communication is severely constrained, then as a processor generates new nodes, it queues them locally for expansion and processes them alone as soon as it can (in an order dependent on its search strategy). Only when a processor is idle and has no nodes queued for expansion does it communicate with other processors to acquire new nodes. We call this *local queuing*. The result is a local approximation to one of the familiar uniprocessor search strategies.

It is possible to impose a global search strategy on a distributed processor by transmitting all nodes ready for expansion to a central repository. A global evaluation function can then order the nodes, and idle processors can remove them (in order) from the repository. We call this *global queuing*. Unfortunately, it can lead to bottleneck and reliability problems. In addition, when communication costs are high, it can lead to lower speedups than local queuing (see Appendix A). The main advantage of global queuing is that it offers the potential

for ensuring that the *best* nodes are expanded first because the evaluation function has a global perspective. When local queuing is used, other measures must be taken to approximate a global perspective. This is an example of the general problem of achieving coherent behavior in a system that uses distributed control. Distributed control is necessary if the advantages of distributed problem solving are to be achieved – but it leads directly to a problem in maintaining global coherence.

Better approximations to global strategies are obtained at the price of interprocessor communication. The intent of a best-first search in a uniprocessor, for example, is to select the most appropriate node for expansion at any given time. If interprocessor communication is severely constrained, then an individual processor can only select the best of the nodes that it has stored locally; and none of these nodes may be the overall best node to be expanded. If the processors can communicate mere extensively with each other, then several of the overall best nodes can be concurrently selected for expansion by separate idle processors. We will see how this is done with the contract net protocol in the next section.

4 Example: The N Queens Problem

The goal of the N Queens problem is to place N queens on an N x N chessboard in such a way that no two are on the same row, column, or diagonal. We discuss one possible implementation of this problem as a simple introductory example of the issues that arise in an application of distributed problem solving. Section 4.1 shows sample messages transmitted by processors during the solution of the problem.

The processor at which the problem is started (the *top-level* processor) begins with an empty board. It generates N subtasks, each of which corresponds to a *partial board* with 1 queen in the first column and in a different row for each subtask. These subtasks are announced. Bids are submitted by other idle processors. Successful bidders are awarded contracts for the task of extending the partial boards to completion. The top-level processor is the manager for this task. (It is now free to become a contractor for future subtasks.)

This process is continued recursively for each column of the board; that is, the contractors trying to extend partial boards (here, with 1 queen already placed) generate independent subtasks by placing a queen in the next column (here, the second column) under the no-capture constraint. They then distribute the subtasks (and take on the role of manager for them).

There is thus only one type of task for all processors—extension of a partial board. When a processor node places the N^{th} queen, and thus has a complete solution to the problem, it reports to its manager. (Similarly, when a processor cannot further extend a partial board, it reports to its manager.) Further reports ripple upward to the top level and the search terminates when some pre-specified number of solutions has been compiled; that is, when any manager has received the required number of solutions, it terminates any outstanding subtasks within its span of control and reports to its own manager. This manager in turn terminates outstanding subtasks, and so on. Ultimately, the top-level node reports the solutions to the user.

The task abstraction of each task announcement specifies the type of task to be executed and the present *state* of the task, relative to the goal state (in this case, the number of queens that have already been placed on the partial board). The number of queens placed gives a potential contractor a method for ranking announced tasks in order to select a task for submission of a bid. It is used by processors in this example to effect an approximation to the desired global search strategy. A breadth-first strategy, for example, is implemented by ranking boards that have a small number of queens placed higher than those that have a larger number of queens placed. Bids are submitted first for these boards, and they are therefore generally executed before the others.[2]

We pointed out earlier that one of the problems associated with distributed control is approximation of the global perspective that enables a uniprocessor to select the best nodes for expansion at any time. This problem is handled in a contract net as follows: Each processor listens to all task announcements and maintains a list of recent announcements. When a processor goes idle, it selects, according to its own criteria, the current optimum task for which to submit a bid from among the tasks contained in its list. Each processor therefore has a kind of *window* through which to view the currently available tasks. This window lends a more global character to the search strategy because node selection is based on information received from a number of processors. The cost is local storage (for the list of tasks) and communication (to gain information about tasks available from other processors).

Two possible eligibility specifications are shown. The first is a null specification. The assumption here is that all processors have the necessary procedures for executing the *extend-board* task. A bid then simply indicates that a processor is willing to execute the announced task, and the contract is awarded to the first bidder. In the second case, the eligibility specification names the required procedures. The assumption here is that not all processors are pre-loaded with the necessary procedures. A potential contractor can submit a bid indicating that it needs the procedures to execute the task. In this case the contract is awarded to the first processor that has the procedures, or, in the absence of any such bidders, to the first bidder. This

[2] It is of course possible to execute different search strategies at different processors.

is an example of dynamic transfer of knowledge. (See **[Smith, 1978]** for a more extensive discussion.) A simple award strategy (i.e., award to the first bidder) is possible for this problem because any processor with the procedures has the capability to execute the task.

4.1 Sample Messages

<The processor given responsibility for the top-level task issues messages of the following form as it generates the first subtasks.>

To: * <"*" indicates a broadcast message.>
From: 1
Type: TASK ANNOUNCEMENT
Contract: 1

Task Abstraction:
TASK TYPE EXTEND-BOARD
BOARD QUEENS 1

Eligibility Specification:
NIL <or> PROCEDURE NAME EXTEND-BOARD

Bid Specification:
NIL

To: 1 <Idle processors respond.>
From: i
Type: BID
Contract: 1

Node Abstraction:
NIL <or> REQUIRE PROCEDURE NAME EXTEND-BOARD

To: i <To the successful bidder.>
From: 1
Type: AWARD
Contract: 1

Task Specification:
BOARD SPECIFICATION (...)
PROCEDURE NAME EXTEND-BOARD CODE (...)
<If required.>

To: k <Eventually, messages like
From: q this are transmitted.>
Type: REPORT
Contract: j

Result Description:
SUCCESS
BOARD SPECIFICATION (...)
<or>
FAILURE

To: n <When enough solutions have been
From: m accumulated by a manager, it
Type: TERMINATION sends messages like this
Contract: i to its contractors.>

5 Summary

We have shown the use of the contract net protocol in the solution of a search problem. The negotiation process is particularly simple for this problem and a minimal amount of information needs to be transferred between processors. Consequently, only a degree of the power of the approach is demonstrated. The main use of the protocol in this example is to make connections between processors for reliable distribution of the processing load and communication of results. Processors are efficiently used because they can take on multiple roles: A processor that has generated all 1-queen extensions to the current board and distributed them to other processors (contractors) need only deal with reports occasionally (in its role as manager). It is therefore free to act as a contractor for other tasks. The result is that no processors remain idle as long as there are tasks to be executed. Furthermore, processors are able to obtain the procedures necessary to execute tasks as part of the negotiation process. Finally, the explicit manager-contractor links assist in rapid local pruning of the search space (via termination messages) when a sufficient number of solutions has been found. Each manager can directly terminate the execution of subtasks being executed by its contractors as soon as it becomes aware that the results are no longer required. The net does not have to wait for reports to reach the top-level processor before subtasks are terminated.

We have demonstrated the utility of negotiation as an approach to the problem of maintaining global coherence in a system that uses distributed control. The problem is by no means solved, however, and is a focal point for further research. One of the extensions currently under examination is to have processors listen more carefully to the message traffic around them. At present, only task announcements are examined by all processors. It may prove beneficial for other messages (e.g., bids, awards, and reports) to be subjected to the same scrutiny. It may, for example, lead to more informed bid and award strategies.

Appendix A

Distributed Search Analysis

We derive bounds on the performance that can be expected from distributed search of regular trees. Although we only explicitly consider regular trees in this analysis, the results are easily extended to a more general class of trees – those that are compositions of regular trees. (See **[Smith, 1978]** for details.)

A.1 Speedup

The total number of nodes, n, in a regular tree structure with depth d and branching factor b is,

$$n = (b^{d+1} - 1)/(b - 1). \quad (1)$$

The number of tip nodes, n_t, in such a tree is

$$n_t = b^d. \quad (2)$$

The time required for the search is the most appropriate measure of performance for a distributed processor. The traditional uniprocessor measure of number of nodes examined is still a valid measure of the power of the search strategy, but is insufficient to capture the effect of multiple processors.

A.1.1 Uniprocessor Search

The search time divides into two components: the time to expand a node, t_e, (i.e., the time required to generate all successor nodes of a node), and the time to select a new node for expansion, t_s. The time to expand a node can be rewritten in terms of the time required to generate a single successor node, t_g, as follows,

$$t_e = b \cdot t_g. \quad (3)$$

The minimum time to find one goal node in a regular tree is achieved if the tree is searched in a depth-first fashion and no false paths are explored. Under this assumption, the uniprocessor search time, t^u_{min}, is,

$$t^u_{min} = ((d - 1) \cdot b + 1) \cdot t_g + (d - 1) \cdot t_s. \quad (4)$$

We have assumed that a node is completely expanded (i.e., all successor nodes are generated) before a new node is selected for expansion, and that the goal node can be recognized as soon as it is generated. We do not consider search strategies where only some of the successors of a node are generated before a new node is selected for expansion. (See **[Smith, 1978]** for treatment of this type of strategy.)

The maximum uniprocessor time, t^u_{max}, is achieved when exhaustive search of the tree must be performed before the goal node is found. In this case, the search time is,

$$t^u_{max} = (n - 1) \cdot t_g + (n - 1 - n_t) \cdot t_s. \quad (5)$$

A.1.2 Distributed Processor Search

We assume the search strategy is as presented in Section 3; that is, a node is distributed for expansion by another processor as soon as it is generated; there is thus no time required for selection of nodes and the search time depends on the time to generate a successor node, t_g, and the time to distribute a node to another processor, t_c. We assume that the t_c cost must be incurred any time the expansion of a node is started by a processor, even if the node was generated by that processor. This is the case if global queuing is used. It leads to a somewhat pessimistic estimate for search time (and therefore speedup) but simplifies the analysis. We will later drop this assumption.

The minimum time for a distributed processor to find a single node in a regular tree, t^d_{min}, is,

$$t^d_{min} = d \cdot t_g + (d - 1) \cdot t_c. \quad (6)$$

The maximum time, t^d_{max}, is given by,

$$t^d_{max} = d \cdot b \cdot t_g + (d - 1) \cdot t_c. \quad (7)$$

This is equivalent to the time required to expand the nodes that border the tree on one side.

A.1.3 Comparison

The speedup for exhaustive, or maximum, search, S_{maxe}, is given by,

$$S_{maxe} = t^u_{max} / t^d_{max} \quad (8)$$

Note that S_{maxe} is not the maximum attainable speedup for a regular tree. It is, however, a convenient measure for comparison. We will later derive the address of the tip node for which the maximum speedup is attained.

Note also that as the selectivity of the search strategy is augmented, thus diminishing the need for exhaustive search, the advantage of concurrent computation is also diminished.

In order to draw some simple conclusions from the S_{maxe} equation, we will assume that $t_s \ll t_g$. Under this assumption,

$$S_{\max e} \approx (n - 1)/(d \cdot b + (d - 1) \cdot (t_c / t_g)). \quad (9)$$

t_c / t_g is a measure of the coupling between processor nodes for the distributed search problem. We call this ratio the *processor-coupling-factor*, C_p. (Note that it depends on both the characteristics of the task and the characteristics of the distributed processor.) Thus, rewriting (9), we have,

$$S_{\max e} \approx (n - 1)/(d \cdot b + (d - 1) \cdot C_p). \quad (10)$$

Figure A.1 shows the variation in S_{maxe} as a function of C_p for a regular tree of branching factor 3 and depth 6. The cost of a mismatch between the task grain size and the communications characteristics of the distributed processor is apparent: Loose-coupling must be ensured by a proper match of task grain size to distributed processor communications characteristics if a significant speedup is to be achieved.

Figure A.2 shows the maximum speedups attainable for exhaustive search of three regular trees, of branching factor 2, 3, and 6, as a function of depth, under the

assumption that $C_p = 0$. Also shown is S_{min}, obtained by comparing the minimum time for a distributed processor to search a regular tree, t^d_{min}, with the minimum time required for a uniprocessor, t^u_{min}, assuming that all successors of a node must be generated before a new node can be selected for expansion.

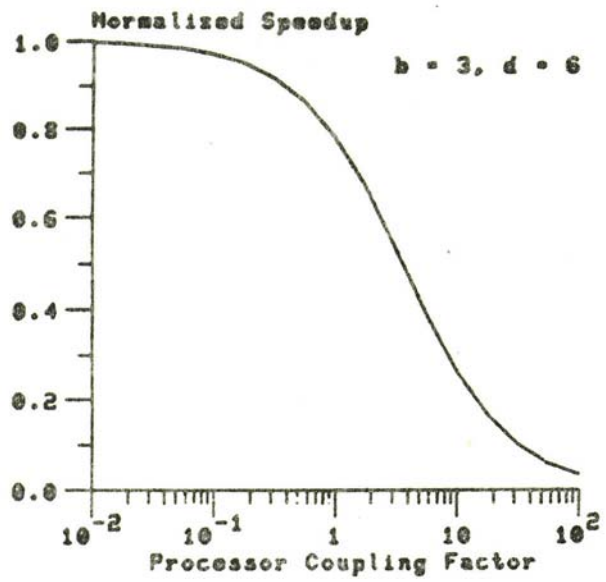

Figure A.1. Speedup And Coupling

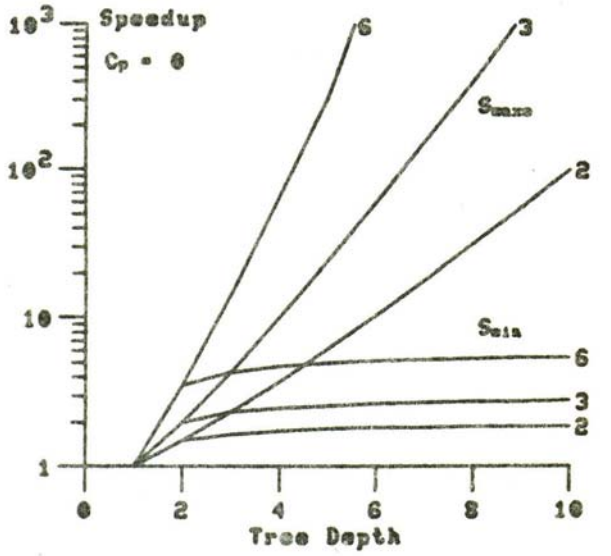

Figure A.2. Speedup For Three Regular Trees

A.2 Processor Requirement

We derive lower and upper bounds on the number of processors, P_{max}, required in a distributed processor to obtain the maximum speedup for the exhaustive search of a regular tree. We assume that $t_c = 0$, and that $t_g = 1$.

As a lower bound, it is apparent that at least S_{maxe} processors are required to achieve a speedup of S_{maxe}. This is overly optimistic because it assumes that all processors are fully utilized throughout the period of the computation. Near the start and end of the search, however, very few processors are in use.

In order to improve the estimate, consider the rate at which processors are pressed into service as the search progresses. The number of new tasks generated at each successive time unit in the search of a tree of infinite depth and branching factor b is given by the following generalized Fibonacci series of order b,

$$P^*_j = (P^*_{j-1} + P^*_{j-2} + \ldots + P^*_{j-b}). \qquad (11)$$
$$j = 1, 2, 3, \ldots$$
$$P^*_j = 0, \quad j < -1.$$
$$P^*_{-1},\ P^*_0 = 1.$$

where the "*" superscript is used to indicate that the series is written for a tree of infinite depth. To account for the finite depth of the trees of interest, the equation can be modified as follows. Observe that whenever a processor reaches a tip node in the tree, the effect is to prune a subtree from the infinite tree. This pruning begins after d time units. We can account for the pruning of these subtrees by subtracting Fibonacci series, that start at times when processors reach tip nodes, from the original series. The number of series to be subtracted at each time instant corresponds to the number of tip nodes reached at that time instant. The number of tip nodes reached at each instant of time (starting at the d^{th} time instant, when the first tip node is reached, to the $b \cdot d^{th}$ time instant, when the search is completed) is given by,

$$k_j = C_b(d, j-d). \qquad (12)$$
$$j = d, d+1, d+2, \ldots, b \cdot d.$$
$$k_j = 0, \quad j < d, \quad j > b \cdot d.$$

where $C_m(n, k)$ is the coefficient in the n^{th} row and the k^{th} column of the m-arithmetic triangle. In general, the $C_m(n, k)$ obey the equation,

$$C_m(n,\ k)\ =\ C_m(n\text{-}1,\ k)\ +\ C_m(n\text{-}1,\ k\text{-}1)\ +\ \ldots\ +\ C_m(n\text{-}1,\ k\text{-}m\text{+}1). \qquad (13)$$

$$0 \le k \le n \cdot (m\text{-}1),\quad n \ge 0.$$

$$C_m(0,\ 0)\ =\ 1$$

$$C_m(1,\ k)\ =\ 1,\quad 0 \le k \le m\text{-}1.$$

$$C_m(1,\ k)\ =\ 0,\quad k\ \ge\ m.$$

Thus the number of processors required, P_j, is given by,

$$P_j^*\ =\ P_j^*\ -\ k_d \cdot P_{j\text{-}d}^*\ -\ k_{d+1} \cdot P_{j\text{-}(d+1)}^*\ -\ \ldots\ -\ k_{b\cdot d} \cdot P_{j\text{-}(b+d)}^*. \qquad (14)$$

$$j\ =\ 0,1,2,\ \ldots,\ b \cdot d.$$

And an upper bound on the number of processors required is given by,

$$P_{max}\ =\ MAX(P_j). \qquad (15)$$

$$0 \le j \le b \cdot d.$$

This estimate is an upper bound because of the assumption that nodes cannot be queued for later expansion, but instead must be expanded as soon as they are generated. This is not generally required because of the lower demand for processors as the search nears completion.

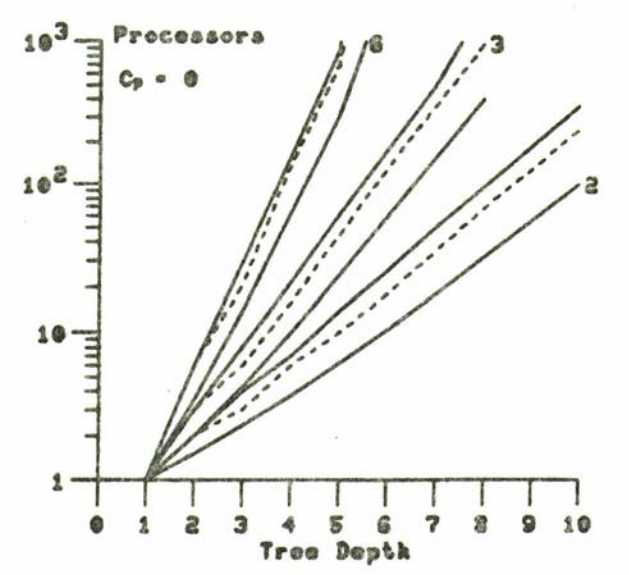

Figure A.3. Processor Requirement

Figure A.3 shows the two bounds for the required number of processors for exhaustive search of the same three trees as used in Figure A.2. Also shown (by dashed lines) is the actual number of processors required to achieve the maximum exhaustive search speedups (as determined by simulation).

Figure A.4 shows the increase in normalized speedup for a tree of branching factor 3 and depth 6 as the number of processors is varied. Also shown is the corresponding decrease in efficiency (i.e., speedup per processor). This is a conservative estimate of efficiency, in that it includes processors that stand idle near the start and end of the search. These processors might be applied to another top-level problem during this time in a general-purpose distributed processor.

We noted earlier that the speedup estimates for distributed processor search are slightly pessimistic, because of the assumption that the cost t_c is *always* incurred when a processor acquires a node for expansion. In Figure A.5, we show the effects of dropping this assumption. The figure compares the possible speedups for varying numbers of processors on a tree of depth 6 and branching factor 3 for both the global queuing of nodes to be expanded and the local queuing of such nodes. The speedups are normalized to that attainable with a global queuing strategy. A small number of sample points are marked with symbols to allow the reader to distinguish between the curves.

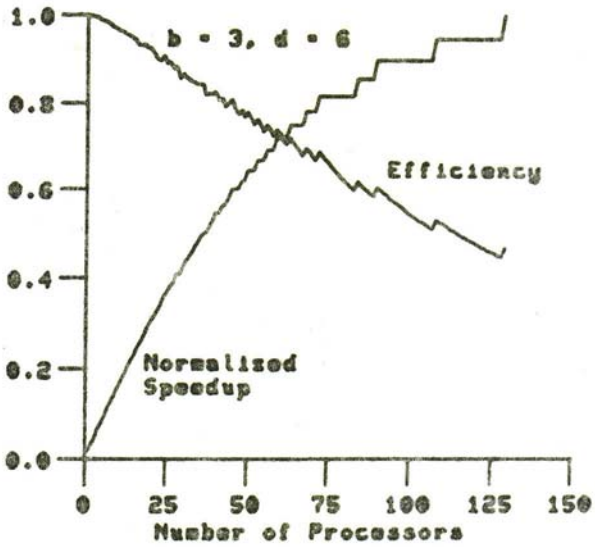

Figure A.4. Speedup And Efficiency

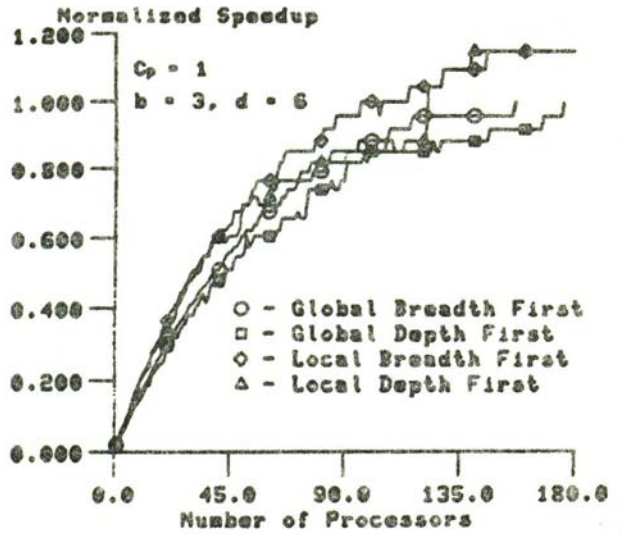

Figure A.5. Speedup And Queuing Strategy

Local queuing strategies are useful when C_p is high. In Figure A.5, $C_p = 1$. We see a small improvement for local queuing in each case.

For further comparison, two selection strategies have been used for the figure: breadth-first and depth-first. We see that a breadth-first strategy leads to slightly better speedups than a depth-first strategy, mainly because tasks get distributed to idle processors earlier in the search.

A.3 The Maximum Speedup

We now derive the *address* of the tip node at which the maximum speedup is attained. As in the previous section, we assume that $C_p = 0$, $t_c = 0$, and $t_g = 1$.

We can write the *address* of a tip node, a_k, as follows,

$$a_k = k + n - n_t. \qquad (16)$$
$$0 \le k \le n_t.$$

where k is the *index* of the tip node. a_k is also the number of time units required by a uniprocessor to reach the tip node with that address, using a breadth-first search algorithm, under the above assumptions.

The number of time units, t_k, required by a distributed processor to reach the node with address a_k is given by,

$$t_k = d + \sum_{j=0}^{d-1} w_j. \qquad (17)$$

where $w_j = \{0, 1, ..., b-1\}$, $0 \le j \le d-1$ are the values of the bits in the b-ary representation of the index, k.

Hence, the address of the node for which the maximum speedup is attained, a^s_{max}, is the a_k that maximizes a_k / t_k.

References

[Buchanan, 1978]
B.C. Buchanan and T.M. Mitchell, Model-Directed Learning Of Production Rules. In D.A. Waterman and F. Hayes-Roth (Eds.), P*attern-Directed Inference Systems*. New York: Academic Press, 1978, pp. 297-312.

[Carhart, 1976]
R.E. Carhart and D.H. Smith, Applications Of Artificial Intelligence For Chemical Inference XX. Intelligent Use Of Constraints In Computer-Assisted Structure Elucidation. *Computers In Chemistry*, Vol. 1, 1976, p. 79.

[Imai, 1979]
M. Imai, Y. Yoshida, and T. Fukumura, A Parallel Searching Scheme For Multiprocessor Systems And Its Application To Combinatorial Problems. *IJCAI6*, 1979, pp. 416-418.

[Smith, 1978]
R.G. Smith, *A Framework For Problem Solving In A Distributed Processing Environment*. Ph.D. Dissertation, STAN-CS-78-700 (HPP-78-28) Dept. of Computer Science, Stanford University, December 1978.

[Smith, 1979]
R.G. Smith, The Contract Net Protocol: High-Level Communication And Control In A Distributed Problem Solver. *Proceedings of the First International Conference On Distributed Computer Systems*, October 1979, pp. 185-192.