RESULTS ON
DISTRIBUTED PROCESSOR SCHEDULING
WITH LIMITED MEMORY
by
Timothy A. Gonsalves
June 1978

Technical Report # 7805

This work was supported by the NSF principally under
Grant # MCS-77-04364 and also under Grants # MCS-72-0351
and # MCS-76-07683.
ABSTRACT.

This paper is a study of scheduling on a 2-processor distributed system when one processor has a limited memory. A program is assumed to consist of a set of sequentially executing modules and is represented by Stone's graph model. It is desired to assign these modules to the processors so as to minimize interprocessor communication while taking advantage of specific capabilities of the two processors. This optimization problem is NP-complete. It is shown that the corresponding 'absolute approximation' problem is as hard. This motivates the development of two polynomial-time heuristic algorithms for the problem. For the class of constant degree graphs it is shown that the heuristics can be useful scheduling tools. Statistics are presented to support the conjecture of Rao, Stone and Hu that use of the 'inclusive-cuts graph' can appreciably simplify this scheduling problem. Finally asymptotic upper and lower bounds on the expected cost of the optimum assignment are derived for the class of constant degree graphs.
1. **Introduction.**

Distributed computing has received increasing attention in the recent literature due to its potential for enhanced flexibility, integrity and performance (Agarwala and Larsen [1976], Enslow [1978], Jensen [1978], van Dam et al. [1974], Michel and van Dam [1977], Foley [1976], Stone [1977, 1977a, 1978], Rao et al. [1977]). Here we consider systems with intermediate coupling between processors, using the terminology of Fuller and Siewiorek [1973]. During program execution control passes between processors to take advantage of specific capabilities of particular processors. For the 2-processor case Stone [1977] uses a network model and network-flow algorithm to solve the problem of static distribution of program modules over processors while minimizing total cost. Two types of costs are considered: costs of execution of individual modules and interprocessor communication costs associated with transfer of control between processors. Rao et al. [1977] show that if one processor has a limited memory capacity the assignment problem is NP-complete. They present a technique which can appreciably reduce the problem size in many instances.

This paper presents results on the complexity and cost of 2-processor scheduling with limited memory. We first show that finding a sub-optimum assignment that incurs at most a constant additional cost is as hard as finding the
optimum assignment. This motivates us to present two polynomial-time heuristic algorithms for the limited-memory problem. Investigation of their performances leads us to obtain asymptotic bounds on the expected cost of the optimum assignment for a class of problem instances.

Section 2 reviews the network model for a 2-processor system and prior work on limited-memory scheduling. In section 3 we prove a result on the complexity of approximation algorithms for limited-memory scheduling. The heuristics and simulation results are presented in section 4. In section 5 we establish bounds on the cost of the optimum assignment and discuss the significance of these results. We summarize our results in section 6.
2. **2-PROCESSOR SCHEDULING.**

We now review Stone's [1977] work on 2-processor scheduling. Next we outline the reduction technique developed by Rao et al. [1977] for 2-processor scheduling with limited memory.

2.1 **2-PROCESSOR SCHEDULING WITHOUT CONSTRAINTS.** (Stone [1977])

A program is assumed to consist of a number of modules. Modules can communicate with each other through a CALL mechanism with parameter passing. Program execution is strictly sequential. Some modules are free to be executed on either processor while others must be assigned to a particular processor. The latter case arises when a module requires resources such as floating-point hardware or a graphics display which are available on only one processor. Each module has a non-negative cost of execution on each processor. If a module cannot be assigned to a processor the corresponding cost is set to infinity. Since the processors are coupled by an intermediate-speed communications link intermodule references between modules on different processors can incur a non-negative communication cost. This cost is assumed to be symmetric i.e. for modules A and B the communication cost is the same whether
A is assigned to processor $P_1$ and B to processor $P_2$ or vice-versa. Inter-module references between modules on the same processor are assumed to have negligible cost. The problem is to find an assignment of modules to processors that minimizes the total cost of execution and communication. There is no limit on the number of modules that can be assigned to either processor.

The costs considered can be a function of any variables of interest to the user such as elapsed time or monetary costs. The costs can be estimates or can be based on previous runs. The assignment is static for the life of the program.

Stone models the problem by a processor-flow graph and uses a network-flow algorithm to solve it. An example graph is given in Fig. 2.1. The graph has one node for each module (A, B, C, D) and one node for each processor (S, T). The weight on the edge between any pair of module-nodes is the communication cost if they are assigned to different processors. The weight on the edge between S and any module-node M is the execution cost of module M on T. Similarly the weight on edge (M, T) is the execution cost of M on S.

Every cutset of the processor-flow graph which separates the processor-nodes uniquely defines an assignment of modules to the two processors. Cutset I in Fig. 2.1 corresponds to the assignment of \{A, C\} to S and \{B, D\} to T. The cost of the assignment is equal to the weight of the cutset. Hence
Fig. 2.1 Processor-Flow Graph.
the problem of finding the minimum cost assignment is equivalent to the problem of finding the minimum weight cutset (mincut) between the processor-nodes in the processor-flow graph. Using the Max-flow, Min-cut Theorem of Ford and Fulkerson [1962] this can be done in time $O(n^2e)$, where $n$ is the number of nodes and $e$ the number of edges in the graph (Dinic [1970], Karzanov [1974], see Even [1976] for an exposition of the Dinic-Karzanov algorithm).

2.2 **SCHEDULING WITH LIMITED MEMORY.** (Rao et al. [1977])

In this section we introduce the constraint that one processor, say T has a limited memory capacity while the other, S has an infinite memory. The size of each program module is known. A feasible assignment is one in which the total size of the modules assigned to T is no greater than its memory capacity. The problem is to find the minimum cost feasible assignment.

Since the 2-processor limited memory scheduling problem is NP-complete it is unlikely that there exists a polynomial-time algorithm for the problem. An exponential-time algorithm is computationally practicable only for small instances of the problem. Rao et al. reduce a processor-flow graph to an inclusive-cuts graph (ICG). The two are equivalent with respect to the minimum feasible assignment. The size of the ICG can be appreciably smaller than that of the processor-flow graph. In section 4.3.2 we present some
statistical evidence for this. In addition there is a partial ordering on the nodes in the ICG. This ordering and the reduction in size can greatly reduce the number of cuts to be examined in an exhaustive search for the minimum feasible cut. Thus construction of the ICG can substantially increase the domain of practicably solvable instances of the problem.

Fig. 2.2b shows the ICG of the example processor-flow graph in Fig. 2.2a. The processor-flow graph is partitioned into disjoint subsets each of which maps onto a single node in the ICG. The minimum cost feasible assignment assigns all the nodes in a subset to one processor. Informally, there is a relatively high degree of communication between nodes in the same subset, i.e., the subset of nodes forms a locality of reference. Thus the ICG can be no larger than the processor-flow graph and may be smaller.

Associated with each node M in the ICG is the weight of the mincut M(S) reassigning the node M to S. Associated with S is the weight of the mincut between S and T. (Note that the weight of a cut in the ICG is the weight of the corresponding cut in the processor-flow graph).

The directed edges of the ICG form paths from T to S. These define a partial ordering on the nodes. Considering any cut in the ICG, we see that if a node M is assigned to S (T) by the cut then every node on every path between M and S (T) is also assigned to S (T). Violating this partial
Fig. 2.2a Processor-flow Graph.

\[ S = \{S, A\} \]
\[ B = \{B\} \]
\[ C = \{C\} \]
\[ D' = \{D, E, F\} \]
\[ G = \{G\} \]
\[ T = \{T\} \]

Fig. 2.2b Inclusive-cuts Graph.
ordering cannot result in a cut of lower value.

Thus construction of the ICG may reduce the number of cuts that need be considered to find the minimum feasible cut. The processor-flow graph of Fig. 2.2a has $2^7 = 128$ possible cuts. Considering the ICG of Fig. 2.2b we have $2^4 = 16$ possible assignments. If we consider the partial ordering we need evaluate only 6 of these assignments to find the minimum feasible assignment.

In this section we have reviewed Stone's graph model and solution procedure for 2-processor scheduling. We have then presented a brief description of limited memory 2-processor scheduling and the inclusive-cuts graph.
3. **COMPLEXITY OF APPROXIMATION ALGORITHMS FOR LIMITED MEMORY SCHEDULING.**

Here we prove that the problem of finding an approximate solution to the limited memory scheduling problem with the cost guaranteed to be within a fixed constant of the cost of the optimum solution is as hard as the problem of finding the optimum solution. In the terminology of Horowitz and Sahni [1977], the absolute approximation 2-processor limited memory scheduling problem is NP-hard. Hence, if we want an efficient algorithm we may have to accept a poor performance in some instances. This motivates the development of efficient heuristic algorithms for the problem.

We introduce some notation and definitions (Horowitz and Sahni [1977]). The abbreviation 2-LMS denotes the 2-processor limited memory scheduling problem. Let A be
an algorithm which generates a feasible solution to every instance I of an optimization problem P. Let \( F^*(I) \) be the value of the optimum solution to instance I of P and \( F'(I) \) the value of the solution found by A.

A is an exact algorithm for P if and only if for every instance I of P, \( F'(I) = F^*(I) \).

A is an absolute approximation algorithm for P if and only if for every instance I of P, \( |F^*(I) - F'(I)| \leq k \), for some constant \( k \geq 0 \).

A is an \( \epsilon \)-approximation algorithm for P if and only if for every instance I of P,
\[
\frac{|F^*(I) - F'(I)|}{F^*(I)} \leq \epsilon,
\]
for some constant \( \epsilon \geq 0 \).

We now show that there exists a polynomial-time absolute approximation algorithm for 2-LMS if and only if there exists a polynomial-time exact algorithm for 2-LMS. Thus, since 2-LMS is NP-complete (Rao et al.[1977]) it is unlikely that we can find a polynomial-time absolute approximation algorithm for 2-LMS. Whether or not there exists an \( \epsilon \)-approximation algorithm for 2-LMS is still an open question. We use the proof techniques of Sahni[1975].

**Theorem 3.1**

" There exists a polynomial-time absolute approximation algorithm for 2-LMS if and only if there exists a poly-
nomial-time exact algorithm for 2-LMS."

Proof.

(i) If there exists a polynomial-time exact algorithm for 2-LMS then trivially there exists a polynomial-time absolute approximation algorithm.

(ii) The proof of the converse is by construction. Given any instance I of 2-LMS we construct an instance I’. We show that by applying an absolute approximation algorithm to I’ we can derive an exact solution to I. Thus if we have a polynomial-time absolute approximation algorithm we can construct a polynomial-time exact algorithm.

Assume that there exists a polynomial-time absolute approximation algorithm A for 2-LMS. F’(I) is the value of the solution found by A for instance I and F*(I) is the value of the optimum solution. For all I,

\[ |F^*(I) - F'(I)| \leq k, \text{ for some constant } k \geq 0. \]

Given an instance I of 2-LMS construct I’ by multiplying all costs by \((k + 1)\). Memory requirements of modules are unchanged. Because the feasibility of a solution depends only on memory requirements there is a one-to-one correspondence between feasible solutions to I and I’. Further, the value of any solution to I’ is \((k + 1)\) times the value of the corresponding solution to I. Therefore there is also a one-to-one correspondence between optimum solutions to I and I’.
Let $F(I)$ be the cost of any solution to $I$ and let $F(I')$ be the cost of the corresponding solution to $I'$.

Assuming integer costs in $I$, for any two solutions $a$ and $b$ we have:

$$|F_a(I) - F_b(I)| = n, \text{ for integral } n \geq 0.$$  

Therefore,

$$|F_a(I') - F_b(I')| = (k+1) \times n \quad (1)$$

Now apply algorithm A to $I'$. The value of the solution found is $F'(I')$. By definition,

$$|F'(I') - F^*(I')| \leq k \quad (2)$$

From (1) and (2) we conclude that $F'(I') = F^*(I')$.

Thus A always finds an optimum solution to $I'$. This defines the corresponding optimum solution to $I$.

Multiplication of costs by $(k+1)$ increases the computation requirements of A by at most a constant factor of $O(\log k)$. Thus since by assumption A is a polynomial-time algorithm we have obtained a polynomial-time exact algorithm for 2-LMS.

Q.E.D.

The implication of Theorem 3.1 is that it is not likely that there exists a polynomial-time absolute approximation algorithm for limited memory scheduling. Thus we are justified in looking for efficient algorithms which may have a poor performance in some cases.
4. **HEURISTICS FOR LIMITED MEMORY SCHEDULING.**

We now present two similar heuristics for 2-processor scheduling with limited memory. One heuristic uses the ICG while the other uses only the processor-flow graph. Both have polynomial-time complexities in all instances. We first describe the heuristics and then present simulation results for randomly generated constant degree graphs. We discuss the results in this and the next section. Details of the heuristic algorithms and results are given in Gonsalves [1978].

4.1 **INDEX-I: A HEURISTIC BASED ON THE ICG.**

The heuristic described here is a "greedy" algorithm
(Horowitz and Sahni [1977]) which is applied to the ICG of a processor-flow graph. This procedure selects one node at a time to assign to T, the processor with limited memory. This step is repeated until the procedure cannot assign any more nodes to T. The selections are made in accordance with the partial ordering imposed by the ICG. After each selection we evaluate the cost of the assignment. The solution found by the heuristic is the lowest valued of the assignments thus obtained.

At each iteration we select the node with the largest saving in cost per unit of memory. Informally, we consider memory to consist of a limited number of units. At each step we "buy" the object which gives the most return per unit "spent". The saving per unit of memory is the index of the node.

We make the selections in accordance with the partial ordering of the ICG by considering at each iteration only 1 those nodes adjacent to nodes already selected. Initially we consider nodes adjacent to T. Further, we do not consider a node N until all nodes on every path from T to N have been selected. The nodes under consideration at each iteration are called candidate nodes.

The saving in selecting a node is the net reduction in

1 Node i is adjacent to node j if and only if there is an edge in the ICG directed from i to j.
communication and execution costs due to reassigning the node from S to T. This saving may be negative. It can be computed in \( O(m) \) time, where \( m \) is the number of nodes in the ICG. However, the change in communication cost depends on the set of nodes already assigned to T. Thus, at every iteration the saving would have to be recomputed for every node under consideration, increasing the amount of computation required. This is compounded if we are interested in assignments for a range of memory capacities on T.

For efficiency we use an approximation to the saving associated with reassigning a node which is invariant with respect to node assignments. Thus we avoid having to recompute the saving for every node in each iteration.

\[
\text{cost of } M(S) - \max_{(M,N) \in \text{ICG}} \left( \text{cost of } N(S) \right) \]

We define
\[
\text{index}(M) = \frac{\text{cost of } M(S)}{\text{Size}(M)},
\]

where \( P(S) \) is the mincut reassigning node P to S. For every node P in the ICG P(S) has been found during construction of the ICG. Clearly this index is independent of node assignments and the memory capacity of T.

The worst-case time complexity of algorithm INDEX-I is \( O(n^2 e) \), where \( n \) is the number of nodes in the processor-flow graph and \( e \) the number of edges. INDEX-I is not an \( \epsilon \)-approximation algorithm.
We illustrate the heuristic using the graph shown in Fig. 4.1a. Module sizes are given in Table 4.1a. The ICG constructed from the graph is given in Fig. 4.1b. Table 4.1b indicates the calculation of indices. Note that the size of D' is the sum of the sizes of D, E and F since D' is formed by condensing D, E and F.

Initially the list of candidates consists of G. We assign G to T, delete it from the list and insert D'. D' is also deleted from the list, assigned to T and C and B are inserted. The size of the modules assigned to T is now 21. We select the node with larger index, B. B can be accommodated on T so we assign it to T and delete it from the list of candidates. Next we consider C. If we assign C to T the memory capacity of T will be exceeded so we delete C. The list is empty and the algorithm terminates. Table 4.1c shows the assignments found in each iteration with the costs of the assignments. The assignment found in iteration #3 has the lowest cost, 28. This is the solution found by the heuristic. In this case it is also the minimum cost feasible assignment.
Min cut
Cost = 19

Fig. 4.1a Processor-flow Graph.

S = \{S, A\}
D' = \{D, E, F\}

Fig. 4.1b Inclusive-cuts Graph.
Table 4.1a Module Sizes.

<table>
<thead>
<tr>
<th>Module, M</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE(M)</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Memory Capacity of T, MEM(T) = 24

Table 4.1b Indices of Modules.

<table>
<thead>
<tr>
<th>Module, M</th>
<th>PROFIT(M)</th>
<th>SIZE(M)</th>
<th>INDEX(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>16</td>
<td>2</td>
<td>8.0</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>2</td>
<td>4.5</td>
</tr>
<tr>
<td>D'</td>
<td>2</td>
<td>11</td>
<td>0.2</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>10</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 4.1c Intermediate Assignments.

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Modules Assigned to S</th>
<th>Modules Assigned to T</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B,C,D',G</td>
<td>-</td>
<td>43</td>
</tr>
<tr>
<td>1</td>
<td>B,C,D'</td>
<td>G</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>B,C</td>
<td>D',G</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>B,D',G</td>
<td>28</td>
</tr>
</tbody>
</table>
4.2 INDEX-G: A SECOND HEURISTIC.

We now describe a heuristic based on the same principles as INDEX-I but which does not involve construction of the ICG. Assume that we have found a minimum cost assignment using a maxflow, mincut algorithm and that the assignment is not feasible. We condense into $S$ the nodes assigned to $S$ and now consider the reduced graph. Initially we assign the remaining nodes to $S$ and iterate assigning a node to $T$ based on an index defined below.

Assume that we have found an intermediate assignment (cut I in Fig. 4.2). If node $N$ is reassigned to $T$ we obtain cut II. The saving due to the reassignment is the cost of cut I minus that of cut II. This saving may be negative. The index of $N$ is the ratio of this saving to the size of $N$. The heuristic proceeds exactly as the heuristic INDEX-I except that there is no partial ordering on the nodes. Hence at each iteration every node assigned to $S$ in the previous iteration is a candidate for reassignment. Also the indices may change with each reassignment and must be recomputed in each iteration.

The worst-case time complexity is $O(n \cdot \max(n, e))$.

INDEX-G is not an $\epsilon$-approximation algorithm.
INDEX(N) = (cost of cut I - cost of cut II)/SIZE(N).

Fig. 4.2 Indices used in INDEX-G.
4.3 \textit{RESULTS OF SIMULATION.}

We first describe the method of generation of the input graphs. Then we present results obtained by applying the heuristics to several constant degree graphs.

4.3.1 \textit{GENERATION OF INPUT GRAPHS.}

The \textit{degree} of a node is the number of edges incident on the node. Average degree is equal to \((n-1)\) times the average connectivity for an \(n\)-node graph, where \textit{connectivity} of a graph is the probability of an edge between any pair of nodes.

Constant degree graphs are intended to represent programs which exhibit the locality of intermodule reference which Stone [1978] conjectures might exist in programs written for distributed processor systems. To represent localities we generate \(q\) independent subsets of module-nodes. Each subset has \(m\) nodes and has connectivity \(p_m\), i.e., within each subset we insert edges between pairs of nodes with probability \(p_m\). The edge weights are randomly distributed. Two processor-nodes are added and each is connected to every module-node. These edge weights also are randomly distributed. The total number of nodes in the graph,

\[
    n = q \cdot m + 2
\]

The average degree of each module-node,

\[
    d_m = p_m \cdot (m-1) + 2, \text{ independent of } n.
\]
The parameters used were $p_m = 0.8; \ m = 10; \ q = 2, 3, 4, 5, 6, 7, 8$; module communication costs exponentially distributed with mean 20; and execution costs exponentially distributed with mean 25. Thus $n = 32, 42, 52, 62, 72, 82$ and $d_m = 9.2$. Module sizes were exponentially distributed with mean 20. The memory capacity of $T$ was set to half the total size of the modules initially assigned to $T$ by the mincut. Since there do not exist any detailed analyses of programs written for distributed processors the choice of parameters was arbitrary.

4.3.2 RESULTS AND DISCUSSION.

The results obtained for constant degree graphs are summarized in Figs. 4.3a, 4.3b and 4.3c.

We note that INDEX-I finds the optimum solution in 77% of the instances used while INDEX-G does so in only 23%. Further, INDEX-I has a mean error of 2% while that of INDEX-G is 11%. However, INDEX-G is 5 to 12 times faster than INDEX-I.

We now discuss the performance of the heuristics. In an $n$-node graph the number of possible assignments is $2^n$. The heuristics examine at most $n$ assignments. The fraction
Fig. 4.3a  Mean Percentage Error versus n.
Fig. 4.3b Percentage of optimum solutions versus n.
Fig. 4.3c Ratio of CPU times versus n.
of assignments examined, \( n/2^n \) is extremely small for the graphs considered ( \( n \geq 32 \) ). This might lead us to expect that the error in the heuristic solutions would be significant. However, the observed mean errors are 2% and 11% for INDEX-I and INDEX-G respectively. In the remainder of this section and in the next section we discuss this performance.

Considering first INDEX-I, we explain its performance by the properties of the ICG. The number of possible cuts in the ICG can be considerably smaller than the number of possible cuts in the original graph. This is due to the node condensation and partial ordering. For the example in section 2.2.1 the processor-flow graph has 128 cuts while the ICG has only 6. Fig. 4.4 shows the mean reduction ratio as a function of input size for constant degree graphs used in the simulation. Reduction ratio is the ratio of input size to ICG size. On the average there is a reduction by a factor of 6. Together with the partial ordering this greatly reduces the number of possible assignments.

In addition, many of the assignments eliminated by the ICG are ones which partition localities. Such assignments have a high communication cost component and hence a high
Fig. 4.4 Mean Reduction Ratio for Constant Degree Graphs.
total cost. To summarize, use of the ICG can greatly reduce the number of assignments to be considered and the assignments that are considered may have an average cost lower than the average cost of the assignments in the original processor-flow graph. From the reduced number of possibly low cost assignments the use of indices enables INDEX-I to find the optimum solution in 77% of the instances and a near optimum solution in the remainder.

We postpone the discussion of the performance of INDEX-G to the next section.

In this section we have described two heuristics for 2-processor scheduling with limited memory. One uses the ICG while the other uses only the processor-flow graph. We have presented results on their performance on randomly generated constant degree graphs and have explained the performance of INDEX-I on the basis of the node condensation and partial ordering in the ICG.
5. **ASYMPTOTIC ANALYSIS OF LIMITED MEMORY SCHEDULING.**

We first discuss the performance of INDEX-G. This leads us to derive asymptotic bounds on the expected cost of the minimum feasible assignment for constant degree graphs. We then discuss the limitations of our analysis.

5.1 **PERFORMANCE OF INDEX-G.**

Let \( n \) be the number of modules and \( S \) and \( T \) be the two processors. The average degree of each module is \((d+2)\) (each module has on the average \( d \) communication edges and 2 execution edges incident on it). Communication and execution costs are exponentially distributed with means \( \bar{C}_c \) and \( \bar{C}_e \) respectively. For an assignment \( I \), let \( C_I \) be the cost of the assignment. Define the normalized cost of assignment \( I \) as

\[
C_{I,\text{norm}} = \frac{C_I}{n}
\]

In this section \( n \) is assumed large unless otherwise stated.

The number of possible assignments that assign \( n_T \) nodes to \( T \) in an \( n \)-node graph,

\[
N(n_T) = \binom{n}{n_T} = \frac{n!}{(n-n_T)! \cdot n_T!}
\]

(1)

\( N(n_T) \) grows rapidly as \( n_T \) increases. \( N(0) = 1 \), \( N(1) = n \), \( N(2) = n^2/2 \), .... For each value of \( n_T \) INDEX-G examines at most one cut. Hence the fraction of all possible cuts
of a given size that it examines,

\[ f(n_T) = \frac{1}{N(n_T)} \]  \hfill (1a)

Even for the smallest graph size considered, \( n = 32 \), and \( n_T \geq 2 \), \( f(n_T) \) is extremely small.

We next obtain an expression for the expected cost of an arbitrary assignment that assigns \( n_T \) nodes to \( T \). This cost consists of an execution cost component \( C_{I,e} \) and a communication cost component \( C_{I,c} \). For each module \( i \) either edge \( (S,i) \) or edge \( (T,i) \) is included in the cut. Thus the expected execution cost component of cut \( I \),

\[ \bar{C}_{I,e} = n \cdot \bar{C}_e \]  \hfill (2)

The probability of an edge between any pair of module-nodes,

\[ p_m = \frac{d}{(n-1)} \]

Thus each module assigned to \( T \) communicates with on the average \( p_m \cdot n_S \) modules assigned to \( S \), where \( n_S \) is the number of modules assigned to \( S \). This contributes \( p_m \cdot n_S \cdot \bar{C}_c \) to the expected communication cost. Summing over all \( n_T \) modules assigned to \( T \),

\[ \bar{C}_{I,c} = p_m \cdot n_S \cdot \bar{C}_c \cdot n_T \]

Since \( n_S + n_T = n \), we have the expected communication cost as a function of \( n_T \),

\[ \bar{C}_{I,c}(n_T) = \frac{d \cdot \bar{C}_c \cdot n_T \cdot (n-n_T)}{(n-1)} \]  \hfill (3)

Thus the expected cost of an arbitrary cut as a function of \( n_T \),
\[
\bar{C}(n_T) = \frac{d \cdot \bar{c}_c \cdot n_T \cdot (n-n_T)}{(n-1)} + n \cdot \bar{c}_e
\]  

(4)

Since \( n \approx (n-1) \) for large \( n \), we have the normalized cost,

\[
\bar{C}_{\text{norm}}(n_T) = d \cdot \bar{c}_c \cdot \frac{n_T}{n} \cdot (1 - n_T/n) + \bar{c}_e
\]  

(4a)

A plot of \( \bar{C}_{\text{norm}}(n_T) \) against \( n_T/n \) is shown in Fig. 5.1. It is seen to be an inverted parabola with minima at \( n_T/n = 0 \) and \( 1 \) and a maximum of \( d \cdot \bar{c}_c/4 + \bar{c}_e \) at \( n_T/n = 0.5 \).

From (4a) and Fig. 5.1 we might expect that the solutions found by INDEX-G would have \( n_T = 0 \) or \( 1 \) in most cases. Fig. 5.2 shows the mean value of \( n_T \) for different sizes of input graphs for INDEX-G and the minimum feasible assignment found by simulation. We see that in all cases except \( n = 32 \), the mean value of \( n_T \) is greater than 3, increasing to nearly 8 in the case \( n = 82 \). This leads us to hypothesize that the use of indices enables INDEX-G to identify localities of modules and thus perform well.

The mean value of \( n_T \) for the minimum feasible cut is consistently higher than that for INDEX-G. We further hypothesize that the indices enable INDEX-G to identify only a few localities. This would also help explain the fact that the performance of INDEX-G worsens as \( n \) increases (Figs. 4.4a and 4.4b).
Fig. 5.1 Expected Normalized Cost of an Arbitrary Cut.
Fig. 6.2 Mean number of modules assigned to T versus n.
5.2 **ASYMPTOTIC BOUNDS ON THE COST OF LIMITED MEMORY SCHEDULING.**

For constant degree graphs with exponentially distributed edge weights as described in section 5.1 we derive asymptotic bounds on the expected cost of the mincut. We then extend these results to the minimum feasible cut.

To obtain a lower bound we consider only execution costs. The mincut is found by assigning each module to the processor on which it has the lower execution cost. Ties are broken arbitrarily. Refering to Fig. 5.3 let I be the mincut. Clearly, the cost of the mincut,

\[ C_I = \sum_{i=1}^{n} \min(c_{S,i}, c_{T,i}), \]

where \( c_{i,j} \) is the weight on edge \((i,j)\)

\( \{c_{S,i}\} \) and \( \{c_{T,i}\} \) are exponentially distributed with mean \( \bar{c}_e \). Hence \( \{\min(c_{S,i}, c_{T,i})\} \) is exponentially distributed with mean \( \frac{\bar{c}_e}{2} \). Hence the expected cost of the mincut,

\[ \bar{c}_I = n \cdot \frac{\bar{c}_e}{2} \]  \hspace{1cm} (5)

The expected normalized cost,

\[ \bar{c}_{I,\text{norm}} = \frac{\bar{c}_e}{2} \]  \hspace{1cm} (6)
Fig. 5.3 Processor-flow graph with only execution costs.
Since edge costs are independent there is an equal probability of a module being assigned to either processor. Hence the expected number of modules assigned to \( T \),

\[
\bar{n}_T = n/2
\]  

(7)

Now consider the case when communication costs are added. If the mincut changes from \( I \) in Fig. 5.3 then the net execution cost contribution to the mincut will increase. In addition there is the contribution from communication costs. Thus \( n \cdot \bar{c}_e / 2 \) is clearly a lower bound on the expected cost of the mincut.

In every instance the value of the mincut is less than or equal to the value of the cut that assigns all the nodes to one processor (i.e. \( n_T/n = 0 \) or 1). Thus the expected value of the mincut is no greater than

\[
\bar{C}(n) = n \cdot \bar{c}_e \quad (\text{from (4a)}).
\]

Thus \( n \cdot \bar{c}_e \) is an upper bound on the expected cost of the mincut.
For certain cases we can obtain a tighter bound. The expected cost of cut I in Fig. 5.3 when communication costs are considered becomes,

\[ \bar{c}_I = n \cdot \bar{c}_e / 2 + \bar{c}_I, c(n/2) \]
\[ = n \cdot \bar{c}_e / 2 + d \cdot \bar{c}_c \cdot n/4, \text{ from (3) and (5)}. \]

Defining \[ r = \frac{d \cdot \bar{c}_c}{\bar{c}_e} \], the ratio of communication cost to execution cost, we have,

\[ \bar{c}_I = \frac{n \cdot \bar{c}_e}{2} \cdot (1 + \frac{d \cdot \bar{c}_c}{2 \cdot \bar{c}_e}) \]

or

\[ \bar{c}_I = \frac{n \cdot \bar{c}_e}{2} \cdot (1 + r/2) \] \hspace{1cm} (8)

Clearly in every instance the value of the mincut must be less than or equal to that of cut I. Hence the expected cost of the mincut is bounded from above by

\[ \min(\frac{n \cdot \bar{c}_e}{2}(1 + r/2), n \cdot \bar{c}_e) \].

For \( r < 2 \) the first term is less than the second term.

Thus for the expected cost of the mincut we have the inequality,

\[ n \cdot \bar{c}_e / 2 \leq \bar{c}_{\text{min}} \leq \min(\frac{n \cdot \bar{c}_e}{2}(1 + r/2), n \cdot \bar{c}_e) \] \hspace{1cm} (9)

Assume that one processor, say T, has a limited memory. We obtain bounds on the expected cost of the minimum feasible cut \( \bar{c}_{\text{mfc}} \). Clearly, \( \bar{c}_{\text{mfc}} \geq \bar{c}_{\text{min}} \).
Thus \( \overline{c}_{mfc} \geq n \cdot \overline{c}_e / 2 \) \hspace{1cm} (10a)

However, since cut I in Fig. 3.2 may not be feasible, this cut is not an upper bound on the minimum feasible cut. Clearly it is feasible to assign all nodes to \( S \). Thus we have
\[
\overline{c}_{mfc} \leq n \cdot \overline{c}_e \hspace{1cm} (10b)
\]

Under certain conditions we can tighten the bound in (10b). We do this by considering the expected cost of the solution found by a heuristic which we describe below.

5.2.1 **HEURISTIC RANDOM.**

For the constant-degree model with parameters given above, let processor \( T \) have a memory capacity of \( M_T \). Denote the mean of the module memory requirements by \( \overline{m} \) and the total program size by \( M \).

We describe the algorithm RANDOM. Initially RANDOM assigns modules at random to \( S \) until the unassigned modules can be accommodated in the memory of \( T \). Let \( R \) be the cut thus found (Fig. 5.4). Let the set of modules assigned to \( S \) be \( N_{S,R} \) and the cardinality of this set be \( n_0 = |N_{S,R}| \).

Now for \( i = 0, 1, 2, \ldots, (n-n_0) \), RANDOM iterates as follows:

Randomly select \( i \) of the unassigned modules and assign them to \( S \) which now has \( n'_S = (n_0 + i) \) modules. For the remaining \( n_1 = (n-n'_S) \) modules, considering only execu-
Fig. 5.4  Cuts Found by RANDOM.
tion costs, find the mincut. Assign the \( n_1 \) nodes to \( S \) and \( T \) according to this mincut. We have thus found a feasible assignment (cut \( I_i \) in Fig. 5.4). RANDOM now increments \( i \) by 1 and repeats this step, initially assigning to \( S \) the same set \( N_{S,R} \).

The solution found by RANDOM is the cut \( I_i \) with the lowest cost among all the cuts \( I_i \), \( 0 \leq i \leq n-n_0 \).

RANDOM performs at most \( n \) iterations. Each iteration requires \( O(n) \) additions and comparisons. Thus in the worst case RANDOM takes \( O(n^2) \) time and is therefore very efficient.

We now derive an expression for the expected cost of the solution found by RANDOM.

The expected value of \( n_0 \),

\[
\bar{n}_0 = \frac{(M - M_T)}{\bar{m}} \tag{11}
\]

In any iteration,

\[
n'_S = n_0 + i \tag{12}
\]

Hence the size of the subset for which RANDOM finds a mincut,

\[
n_1 = n - n'_S = n - (n_0 + i)
\]

For a given \( i \), the expected value of \( n_1 \),

\[
\bar{n}_1 = n - (\bar{n}_0 + i)
\]

From (7) the expected number of nodes assigned to \( T \),

\[
\bar{n}_T = \bar{n}_1/2 = n/2 - \left( \frac{\bar{n}_0 + i}{2} \right) \tag{13}
\]
From (12) we have the expected value of $n_S'$

$$n_S' = n_0 + i$$  \hspace{1cm} (14')

The cost of cut $I_i$ consists of three components: the total communication cost, the execution cost of the $n_S'$ nodes randomly assigned to $S$ and the execution cost for the $n_1$ nodes for which a mincut was found.

From (3) the expected value of the first component is,

$$\overline{C}_{I_i,c} (\overline{n}_T) = \frac{d \cdot \overline{c}_c \cdot \overline{n}_T \cdot (n - \overline{n}_T)}{n-1}$$

The second component is clearly $\overline{n}_S' \cdot \overline{c}_e$.

From (5) the third component is $\overline{n}_T \cdot \overline{c}_e / 2$.

Thus the expected cost of cut $I_i$,

$$\overline{C}_{I_i} = \frac{d \cdot \overline{c}_c \cdot \overline{n}_T \cdot (n - \overline{n}_T)}{n-1} + \overline{n}_S' \cdot \overline{c}_e + \overline{n}_T \cdot \overline{c}_e / 2$$

Substituting the expressions for $\overline{n}_T$ and $\overline{n}_S'$ from (13) and (14) and since $n=\overline{n}+1$, we have,

$$\overline{C}_{I_i} = \frac{d \cdot \overline{c}_c \cdot (n^2 - (n_0+i)^2)}{4n} + \frac{\overline{c}_e}{2} \cdot (n+n_0+i)$$

Replacing $(n_0+i)/n$ by the variable $\overline{x}$,

$$\overline{C}_I (\overline{x}) = \frac{n \cdot \overline{c}_e}{2} \cdot \left[ (1+\overline{x}) + \frac{r}{2} \cdot (1-\overline{x}^2) \right]$$  \hspace{1cm} (15)

where $r = \frac{d \cdot \overline{c}_c}{\overline{c}_e}$

Normalizing,

$$\overline{C}_{I,\text{norm}} (\overline{x}) = \frac{\overline{c}_e}{2} \cdot \left[ (1+\overline{x}) + \frac{r}{2} \cdot (1-\overline{x}^2) \right]$$  \hspace{1cm} (15a)
Fig. 5.5 shows the variation of $\bar{C}_{I,\text{norm}}(\bar{x})$ with $\bar{x}$ for various values of $r$. For each value of $r$ we plot two curves, one obtained from (15a) and the other obtained by simulation. For each value of $r$ RANDOM was tested on 100 randomly generated constant degree graphs each of size 102 nodes. The analytical and simulation results agree closely.

For any memory capacity $M_T$ on $T$ we have the expected fraction of the program that cannot be assigned to $T$,

$$\bar{f}_0 = \bar{n}_0/n = (M - M_T)/M \quad \text{(from (11)).}$$

The expected costs of the cuts $I_i$ found by RANDOM lie in the interval $[\bar{f}_0 \leq \bar{x} \leq 1]$. The expected cost of the solution found by RANDOM is clearly the minimum of $\bar{C}_I(\bar{x})$ in the above interval. For $r > 2$ this minimum is $n \cdot \bar{c}_e$ (Fig. 5.5) and for $r < 2$ it is $\bar{C}_I(\bar{f}_0)$.

Since the solution of RANDOM is feasible we now have an upper bound on the expected value of the minimum feasible cut,

$$\bar{c}_{\text{mfc}} \leq \min\left(\bar{C}_I(\bar{f}_0), n \cdot \bar{c}_e\right)$$

or,

$$\bar{c}_{\text{mfc}} \leq \min\left(\frac{n \cdot \bar{c}_e}{2} \cdot [(1 + f_0) + \frac{r}{2} \cdot (1 - f_0^2)], n \cdot \bar{c}_e\right) \quad (16)$$

Together with the lower bound in (10a) this yields the inequality,

$$n \cdot \bar{c}_e/2 \leq \bar{c}_{\text{mfc}} \leq \min\left(\frac{n \cdot \bar{c}_e}{2} \cdot [(1 + f_0) + \frac{r}{2} \cdot (1 - f_0^2)], n \cdot \bar{c}_e\right) \quad (16a)$$
Fig. 5.5 Expected value of cuts found by RANDOM.
5.2.2 DISCUSSION

First we consider the expected error in the solution found by RANDOM. Define 
\[ \varepsilon = \frac{|C_{\text{heuristic}} - C_{\text{optimum}}|}{C_{\text{optimum}}} \]

From (16a) we have a bound on the expected error of RANDOM,

\[ \bar{\varepsilon} \leq \min \left\{ \frac{n \cdot \bar{c}_{e} \cdot [(1 + \bar{f}_{0}) + \frac{r}{2} \cdot (1 - \bar{f}_{0}^{2})]}{2}, \frac{n \cdot \bar{c}_{e}}{2} \right\} - \frac{n \cdot \bar{c}_{e}}{2} \]

or,

\[ \bar{\varepsilon} \leq \min \left\{ [\bar{f}_{0} + \frac{r}{2} \cdot (1 - \bar{f}_{0})], 1 \right\} \]

Thus for \( r \geq 2 \), the expected error in the solution found by RANDOM is no more than 100\%. For \( r < 2 \), that is if communication costs are not large compared to execution costs, the expected error may be strictly less than 100\%.

Since RANDOM is very efficient we can always use it in addition to any other heuristic and take as the solution of the composite heuristic the lower of the solutions found independently by RANDOM and the other heuristic of interest. Thus the bound on the expected performance of RANDOM is also a bound for any such composite heuristic.

Consider any heuristic which examines several feasible cuts and chooses as its solution the cut with lowest cost. Assume that one of the cuts examined is always the cut assigning all nodes to \( S \). This cut clearly has expected
value $n \cdot \overline{c}_e$ since it contains only the edges $(i, T)$ with cost $c_{i,T}$, $1 \leq i \leq n$. Hence any such heuristic has an expected error,

$$\overline{\epsilon} \leq \frac{n \cdot \overline{c}_e - n \cdot \overline{c}_e/2}{n \cdot \overline{c}_e/2}$$

or,

$$\overline{\epsilon} \leq 1$$

Thus for constant degree graphs any such heuristic on the average for large $n$ finds a solution that costs no more than twice the optimum solution. This is a characteristic of the two heuristics INDEX-I and INDEX-G.

We note that the results of this section are valid even if there is a particular subset of modules, independent of $n$, which must be assigned to a specific processor due to special capabilities provided only by that processor. This can be seen easily by considering normalized costs of assignments. The maximum cost that these modules together can contribute to any cut is constant. Thus, when normalized, for large $n$ this is vanishingly small.

An implication of this analysis is that, subject to the limitations of the model, the average cost of distributed processing of large programs is at least 50% of the cost of execution of the program on one processor only. In a practical situation where costs may not be known with any degree of precision this saving may not be significant compared to the cost of finding an optimum or near-optimum assignment.
In such a situation the most effective strategy may be to assign to the limited memory processor those modules that are constrained to run on it and the remainder to the other processor.

For programs with $r \leq 2$ and $f_0$ small even a simple heuristic such as RANDOM has a small average error. For such cases it is probably not worthwhile to use complex algorithms which perform marginally better than simple ones.

It is in order here to discuss the limitations of the model and analysis. The analysis is for large $n$ and does not say anything about small programs. The assumption of exponentially distributed costs is in keeping with the intuitively appealing notion that small costs occur more frequently than large costs. Given an arbitrary distribution with costs ranging between $c_{\min}$ and $c_{\max}$ we have the inequality,

$$n \cdot c_{\min} \leq C_{\text{mfc}} \leq n \cdot c_{\max}$$

For particular distributions this range may be narrowed. The assumption that on the average each module communicates with a fixed number of other modules is reasonable for well-structured programs. Modules which are used by a large percentage of other modules can profitably be replicated on both processors since such modules are usually small. A more meaningful discussion of these issues must await the development of a sufficient base of data on programs written
for distributed processor systems.

In this section we have discussed the performance of INDEX-G. This has lead us to derive asymptotic bounds on the expected cost of limited memory scheduling for a class of problem instances. We have discussed the implications and limitations of our analysis.
CONCLUSIONS.

We have first shown that finding an absolute approximation solution to the 2-processor limited memory problem is as hard as finding an exact solution. This has motivated us to develop and test two polynomial-time heuristics. We have shown that INDEX-I performs 9% better than INDEX-G but is much slower. If costs are not known precisely the 9% improvement may not be significant, especially in view of the faster execution of INDEX-G. However, these scheduling techniques are viable only for production programs which are executed often. Over a period of time the recurring saving due to the lower cost of the assignment found by INDEX-I may justify the extra computation required initially. Another case in which INDEX-I may be useful is when we desire to find assignments for a range of memory capacities on T. Since the ICG need be computed only once, the execution time for INDEX-I will increase marginally while that for INDEX-G will increase substantially.

The simulation results have led us to derive asymptotic bounds on the expected cost of the minimum feasible assignment. For constant degree graphs with exponentially distributed costs we have shown that even a simplistic heuristic need have an expected error no greater than 100%. In certain cases the expected error is strictly less than 100%. Simulation closely confirms our analysis. Our results imply that
in some cases the use of complex algorithms for distributed processor scheduling may not be cost-effective. We have examined the limitations of our model and have concluded that subject to formal validation of our model our assumptions are reasonable.

ACKNOWLEDGEMENTS.

I thank Dr. G.S. Rao for his encouragement, advice and many discussions in all phases of this work. My thanks to Bart Sinclair for several fruitful discussions.
REFERENCES.


Michel, Janet and Andries van Dam, [1977], "Evaluation of Performance Improvement in Distributed Processing", draft, Program in Computer Science, Brown University, Providence, RI, 18 pp, Aug. 1977.


