A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion

CHIEN-CHUNG SHEN AND WEN-HSIANG TSAI, MEMBER, IEEE

Abstract — **A graph matching approach is proposed in this paper for solving the task assignment problem encountered in distributed computing systems. A cost function defined in terms of a single unit, time, is proposed for evaluating the effectiveness of task assignment. This cost function represents the maximum time for a task to complete module execution and communication in all the processors. A new optimization criterion, called the minimax criterion, is also proposed, based on which both minimization of interprocessor communication and balance of processor loading can be achieved. The proposed approach allows various system constraints to be included for consideration. With the proposed cost function and the minimax criterion, optimal task assignment is defined. Graphs are then used to represent the module relationship of a given task and the processor structure of a distributed computing system. Module assignment to system processors is transformed into a type of graph matching, called weak homomorphism. The search of optimal weak homomorphism corresponding to optimal task assignment is next formulated as a state-space search problem. It is then solved by the well-known *A** **algorithm in artificial intelligence after proper heuristic information for speeding up the search is suggested. An illustrative example and some experimental results are also included to show the effectiveness of the heuristic search.**

Index Terms — ***A** algorithm, distributed computing systems, graph matching, interprocessor communication, load balancing, minimax criterion, state-space search, weak homomorphism.**

I. INTRODUCTION

DISTRIBUTED computing systems have become more and more attractive and important in recent years due to the advancement of VLSI and computer networking technologies. Distributed computing systems not only provide the facility for utilizing remote computer resources or data not existing in local computer systems but also increase the throughput by providing facilities for parallel processing [1]. Furthermore, the modularity, flexibility, and reliability of distributed computing systems make them attractive to many types of applications. But there are some major problems that prevent widespread use of the distributed computing system [1], [2], [16]. One of the problems is the degradation in throughput caused by the "saturation effect" due to excessive interprocessor communication by data and control messages transferred from one program module to another residing in different processors [2].

Thus, the purpose of task assignment in distributed computing systems is to reduce the job turnaround time and increase the throughput. This can be done by maximizing and balancing the utilization of resources while minimizing the communication between processors [1]. While minimizing interprocessor communication tends to assign the whole task to a single processor, load balancing tries to distribute the program modules of the task evenly among the processors. Therefore, there exists conflict between these two criteria and a compromise must be made to obtain an optimal policy for task assignment.

Several approaches to task assignment in distributed computing systems have been suggested [1]–[12], [16]. They can be roughly classified into three categories, namely, graph-theoretic [3]–[6], mathematical programming [2], [16], and heuristic methods [1]. The graph-theoretic method uses a graph to represent a task, and applies the minimal-cut algorithm to the graph to get the task assignment with minimum interprocessor communication. The mathematical programming approach formulates task assignment as an optimization problem, and solves it with mathematical programming techniques. And the heuristic method provides fast but suboptimal algorithms for task assignment, which are useful for applications where an optimal solution can not be obtained in real time.

Most of the above methods adopt some types of cost function to evaluate the effectiveness of task assignment algorithms [1]–[12], [16]. The most commonly used cost function is defined as the sum of the interprocessor communication cost and the processing cost. But these two types of cost are measured in different units, and it is difficult to give a reasonable meaning to the resulting cost summation.

In this paper, we propose a new task assignment model for distributed computing systems, based on a graph matching approach and a more meaningful cost function for task assignment optimization. Each graph match corresponds to a specific task assignment. Cost values are defined in terms of a single unit, time. Minimization of cost functions is based on a so-called minimax criterion. The proposed model allows easy incorporation of most system constraints encountered in

Manuscript received November 9, 1983; revised May 7, 1984.

The authors are with the Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, Republic of China.

0018-9340/85/0300-0197$01.00

applications. A state-space search method [15] is employed for finding optimal task assignment corresponding to minimum-cost graph matching. Useful heuristic information to speed up the search is also suggested. Optimal solutions guaranteed by the proposed approach are important for non-real-time applications where the resulting assignment will be repeatedly executed on a distributed computing system. And speedup in solution search is useful for real-time applications.

In the remainder of this paper, we describe system assumptions, the cost function, and the minimax criterion in Section II. In Section III, we formulate the graph matching model, and in Section IV, we review the state-space search algorithm and apply it to the search of optimal solutions. An illustrative example and some experimental results are given in Section V, followed by conclusive remarks in the last section.

II. Assumptions, Costs, and Optimization Criterion

Various assumptions made about the distributed computing system considered in this paper are first described in the following.

1) The processors in the system are heterogeneous. That is, a single program module, if executed on different processors, will require different amounts of running time, which specify different degrees of preference of the module on the processors.

2) Nonidentical communication links are used by the processors for message transmission. That is, an identical amount of message, if transmitted through different communication links, will require different amounts of transmission time, which specify different degrees of preference of the message on the links.

3) The processors in the system need not be fully connected. But the link between any two processors is symmetric, i.e., the time to transmit a certain amount of message from one processor to another is identical to that to transmit the same message in the reverse direction.

4) There exists little or no precedence relationship or synchronization requirement among the program modules so that processor idleness is negligible during task execution, provided that module partition has been resolved satisfactorily.

Based on the above assumptions, the cost function and the minimax criterion proposed for task assignment optimization are described in the following.

After a task is partitioned into suitable modules, let $t_p^e(A)$ denote the total time spent for module execution, and $t_p^c(A)$ be the total time for interprocessor communication delay, both in some processor p according to a certain task assignment A. Let

$$t_p(A) = t_p^e(A) + t_p^c(A)$$

which is the total time spent in processor p for task assignment A. We call $t_p(A)$ the *processor turnaround time of* p. This turnaround time is different for each distinct processor. Let

$$t(A) = \max_p t_p(A)$$

which we call the *task turnaround time of* A. It is easy to see that $t(A)$ is the total time required to complete the whole task according to assignment A under the assumption of negligible processor idleness. Therefore, $t(A)$ may be, from the point of reducing total processing time, used as a cost measure for the effectiveness of task assignment A. The smaller $t(A)$ is, the better A is. An optimal task assignment thus may be defined as the one A_o which minimizes the task turnaround time $t(A)$, i.e.,

$$t(A_o) = \min_A t(A) = \min_A \max_p t_p(A).$$

This means that we want to minimize the maximum processor turnaround time, resulting in the so-called *minimax criterion*. $t(A_o)$ will be called the *minimum task turnaround time*.

III. Graph Matching Model

Graphs are of a type of very general data structure for system representation. The graph matching problem is that of finding an efficient algorithm to compare two given graphs to see how well they match in a certain sense. A survey of such problems and the algorithms for solving them can be found in [13], [14]. Based on the purpose or the criterion of matching, there are various types of graph matching. The type we consider in this paper is weak homomorphism which is defined as follows.

Definition 1: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs where V_i $(i = 1, 2)$ are the vertex sets and E_i $(i = 1, 2)$ are the edge sets of G_1 and G_2, respectively. G_1 is weakly homomorphic to G_2 if there exists a mapping (possibly many-to-one) M: $V_1 \rightarrow V_2$ such that if edge $(a, b) \in E_1$, then edge $(M(a), M(b)) \in E_2$, and we say that there is a weak homomorphism from G_1 to G_2.

As mentioned before, a task submitted into a distributed computing system is partitioned into suitable modules and then assigned to processors. Each task thus can be represented by an undirected graph $T = (V_T, E_T)$, which we call the *task graph* where 1) V_T is a set of vertices each of which represents a module of the task; 2) $E_T \subseteq V_T \times V_T$ is a set of edges each of which represents the intermodule communication between the two modules at the ends of the edge.

Similarly, the processors in a distributed computing system can also be represented by an undirected graph $P = (V_P, E_P)$, which we call the *processor graph* where 1) V_P is a set of vertices representing the processors in a distributed computing system; 2) $E_P \subseteq V_P \times V_P$ is a set of edges representing the communication links between processors.

Because two related modules may be assigned to a single processor, we add a self-looping edge to each vertex in the processor graph to follow the weak homomorphism definition, as will be clear in the following discussion.

Fig. 1 shows a typical task graph and Fig. 2 shows a typical processor graph.

For a task assignment A to be acceptable, any two modules a and b which communicate with each other *must* be assigned either to a single processor X or to two processors X and Y with a communication link between them. In terms of graph-theoretic terminologies just defined and considering the task assignment A as defining a mapping M from the task graph T to the processor graph P, we see this requirement means that if edge $(a, b) \in E_T$, then it must be true that edge $(X, X) \in E_P$, or that edge $(X, Y) \in E_P$. Edge (X, X) is always included in the processor graph P as mentioned previously. By Definition 1, this in turn means that there must exist a weak homomorphism from T to P. Therefore, the task assignment problem can be transformed into the problem of finding a weak homomorphism from the task graph T to the processor graph P with minimum task turnaround time. We call such a weak homomorphism the *optimal weak homomorphism from T to P*. Now, we have to discuss how task turnaround time (i.e., the cost value) can be computed for each mapping or weak homomorphism.

Definition 2: Vertex Transformation — Let T and P be the task graph and the processor graph defined previously. If M is a mapping from T to P defined by a task assignment A, then we use $a \rightarrow M(a)$ to denote the assignment of module $a \in V_T$ to processor $M(a) \in V_P$, and call it a vertex transformation.

Let C_p be a nonnegative real-valued cost function defined on all vertex transformations such that $C_p\ (a \rightarrow M(a))$ denotes the execution time of module a on processor $M(a)$.

Definition 3: Edge Transformation — Similarly, we use $(a, b) \rightarrow (M(a), M(b))$ to denote the assignment of module communication between modules a and b to the communication link between processors $M(a)$ and $M(b)$, and call it an edge transformation.

Let C_c be another nonnegative real-valued cost function defined on all edge transformations such that C_c $((a, b) \rightarrow (M(a), M(b)))$ denotes the communication time between modules a and b using the communication link $(M(a), M(b))$. If $M(a)$ is equal to $M(b)$, i.e., if modules a and b are assigned to the same processor, then the cost is defined to be zero. For those edges (a, b) nonexistent in E_T, the costs for assigning them to any communication links are also set zero.

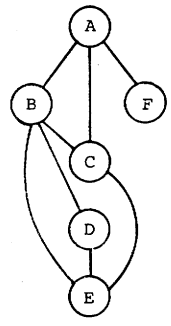
If there are n modules $m_1, m_2, \cdots, m_n$ in a task and m processors $p_1, p_2, \cdots, p_m$ in a distributed computing system, we can use an n by m matrix X to represent the mapping M: $V_P \rightarrow V_T$ where

$$X_{ij} = \begin{cases} 1, & \text{if module } i \text{ is assigned to processor } j; \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the total execution time of all the modules assigned to processor j is

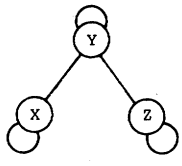
$$PT_j = \sum_{i=1}^{n} [C_p(m_i \rightarrow p_j) \cdot X_{ij}]. \tag{1}$$

Let L be an m by m matrix representing the configuration




```
T = (V_T, E_T) where
V_T = {A, B, C, D, E, F},
E_T = {(A,B),(A,C),(A,F),(B,C),(B,D),
       (B,E),(D,E),(C,E)}.
```

Fig. 1. Task graph.




```
P = (V_P,E_P) where
V_P = {X, Y, Z},
E_P = {(X,Y),(Y,Z),(X,X),(Y,Y),(Z,Z)}.
```

Fig. 2. Processor graph.

structure of the processors in a distributed computing system where

$$L_{ij} = \begin{cases} 1, & \text{if processors } i \text{ and } j \text{ are connected to each other;} \\ 0, & \text{otherwise.} \end{cases}$$

Let R be an n by n matrix representing the intermodule communication relationship of a task where

$$R_{ij} = \begin{cases} 1, & \text{if modules } i \text{ and } j \text{ communicate with each other;} \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the total communication time spent in processor j is

$$CT_j = \sum_{k=1}^{m} \sum_{s=1}^{n} \sum_{t=1}^{n} [C_c((m_s, m_t) \rightarrow (p_j, p_k)) \cdot X_{sj} \cdot X_{tk} \cdot R_{st} \cdot L_{jk}]. \tag{2}$$

In the above equation, X_{sj} and X_{tk} are included to indicate whether m_s is assigned to p_j and whether m_t is assigned to p_k.

According to (1) and (2) the processor turnaround time of processor j is equal to

$$TA_j = PT_j + CT_j. \tag{3}$$

Recalling that the task turnaround time is just the maximum of all the processor turnaround times, we can define the cost of a mapping M defined by a task assignment to be

$$\text{COST}(M) = \max_{1 \le j \le m} (TA_j). \tag{4}$$

According to the minimax criterion, our goal in this paper now is to find an optimal mapping M^* which is a weak homomorphism such that

$$\text{COST}(M^*) = \min_{M} \text{COST}(M).$$

This optimal weak homomorphism M^* not only minimizes the interprocessor communication time but also achieves load

balancing and will be found by the state-space search method described in the next section.

IV. State-Space Search Method

In this section, the problem of finding the optimal weak homomorphism between two graphs will be formulated as a state-space search problem, and the well-known A^* algorithm in artificial intelligence will be used for solving this problem [15]. With the A^* algorithm, not only the optimal weak homomorphism is guaranteed to be found, but the search will also speed up. Note that this approach so far has not been considered in any other investigation on task assignment in distributed computing systems.

In a state-space search problem, each state description is denoted by a *node*. *Operators* applicable to nodes are defined for generating successors of nodes, called *node expansion*. A *solution path* of a search problem is a path in the state space defined by a sequence of operators which leads a *start node* to one of the *goal nodes* [15]. In our case here, a solution path defines the optimal weak homomorphism which is a weak homomorphic mapping M with minimum COST(M). We now formulate the state-space search method as follows.

1) State Description: Let ordered pair set $K = \{(i,x) \mid i \in V_T, \text{ and } x \in V_P\}$ denote the partially developed mapping M corresponding to a tree node n in the search tree. Each ordered pair (i,x) means that module i is assigned to processor x, i.e., $M: i \rightarrow x$ or $x = M(i)$. Let $K_G = \{i \mid (i,x) \in K\}$ and $i_{\max} = \max_{i \in K_G} i$.

2) Initial State: The initial state is $K \neq \emptyset$, the empty set.

3) Operators: An operator adds new *valid* pairs (j,y) to K. The procedure is as follows. First, form a set of candidate ordered pairs

$$D = \{(j,y) \mid j = i_{\max} + 1, y \in V_P\}.$$

Next, check all candidate pairs one by one for validity, using vertex adjacency information in the task graph T and the processor graph P. A candidate pair $(j,y) \in D$ is said to be *valid* if for each $(i,x) \in K$ it is true that if edge $(i,j) \in E_T$, then either edge $(x,y) \in E_P$ or $x = y$. This validity check is necessary to preserve the property of weak homomorphism (see Definition 1). Each candidate pair (j,y) represents a possible additional assignment of module j to processor y, but if j communicates with any other module i already assigned to a processor x (i.e., $i \in K_G$), then j should also be assigned either to processor x or to another distinct processor y which has a communication link to x. Let D' be the set of all valid pairs in D. Then the operator finally updates set K as the union of old K and D'.

Note that the operator is defined in such a way that it always expands the partially developed mapping M (as represented by K) by assigning the module *with next larger index number* $j = i_{\max} + 1$ (see the definition of D above) to each of all the processors y in V_P. It is in this way that developing of partial mapping can be kept in order until all possible module assignments are exhaustively considered (see Fig. 5 for an illustration).

4) Goal State: Any state K with $V_T = K_G$ is a goal state; that is, the search stops when all modules are assigned completely.

The above formulation just offers a search scheme for finding a homomorphic mapping corresponding to a task assignment; it does not include computation of cost values. Next, we use the A^* algorithm described in Nilsson [15] to find optimal weak homomorphisms.

In an A^* algorithm, an evaluation function is used to order nodes for expansion, and is guaranteed to find a solution path optimal in the sense that the path cost is minimized, if the evaluation function is properly defined. More specifically, according to Nilsson [15], if we define an evaluation function as

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the minimum path cost from the start node to node n in the state space, and $h(n)$ is a lower-bound estimate, using any heuristic information available, of the minimum path cost $h^*(n)$ from node n to a goal node, then as long as $h(n) \leq h^*(n)$ for all n (i.e., $h(n)$ is *consistent*), the A^* algorithm using such an evaluation function will be guaranteed to find an optimal solution path in the state space after expanding fewer nodes during the search than any uninformed algorithms. In other words, such a heuristic search algorithm can speed up the search of an optimal weak homomorphism, which is usually time consuming for graphs with large numbers of vertices and edges.

According to the previous formulation of task assignment as the graph matching problem, a node n in the state space denotes a partially developed homomorphic mapping which assigns part of the modules of a task to processors. Thus, we can let the evaluation function at node n in the state space be

$$f(n) = g(n) + h(n) \tag{5}$$

where the value of $g(n)$ is the total minimum cost of the vertex and edge transformations included in the partially developed mapping M corresponding to n, and the value of the heuristic function $h(n)$ is an estimate of the value $h^*(n)$ which is the minimum cost of the vertex and edge transformations required to complete the partial mapping M.

To calculate $g(n)$, we first set up the X and R matrices as mentioned in the previous section for the partial mapping M corresponding to node n, and then use (1), (2), (3), and (4) to calculate the cost of a partial mapping M which defines $g(n)$.

As to the heuristic function $h(n)$ in (5), it can be defined by several different approaches. The simplest way is to set $h(n) \equiv 0$ for all n, and the resulting search is a uniform-cost search [15] which nevertheless is still guaranteed to find an optimal weak homomorphism. To be more efficient in the search, nonzero $h(n)$ values, of course, should be used, and this has been found possible as is discussed in the following.

Let M be the corresponding partial mapping at search tree node n. When using (4) to find the cost of M, try to find the k such that

$$\begin{aligned} \text{COST}(M) &= \text{COST}(TA_k) \\ &= \max_{1 \leq j \leq m} (TA_j). \end{aligned}$$

That is, processor k is currently with the maximum processor turnaround time in the partial task assignment. Let MK be the set of modules assigned to processor k by the partial mapping M, and MK' be the set of all the remaining modules not assigned by M. Then, define $LK = \{(a,b) \mid a \in MK', b \in MK$, and $(a,b) \in E_T$ (edge set of task graph)$\}$ which denotes a set of the edges representing all the communications between the modules in MK and MK', and define $B = \{a \mid (a,b) \in LK\}$ which denotes the set of those unassigned modules in MK' communicating with the modules in processor k. The heuristic information useful for defining $h(n)$ is observed as follows. When the state-space search process goes forward, each module a in B will *have to* be assigned either to processor k or to a processor l connected through a communication link to processor k so that communication between module a and those modules in k can be accomplished. But these two different types of assignments will result in different amounts of partial processor turnaround time being increased at processor k. To keep $h(n)$ as a lower bound of $h^*(n)$, we must find out the smallest amount for each module a in B and then sum up these amounts for all the modules in B as the value $h(n)$, as is calculated in the following algorithm.

Algorithm 1: Computation of $h(n)$. M is the corresponding partial mapping at search tree node n.

Step 1): Find the k such that

$$\text{COST}(M) = \text{COST}(TA_k) = \max_{1 \le j \le m}(TA_j).$$

Step 2): Find the sets MK, MK', LK, and B as defined previously.

Step 3): For each $a \in B$, find $L_a = \{b \mid (a,b) \in LK\}$ which is the set of the modules in processor k communicating with module $a \in B$.

Step 4): Define $AC_a = \min_{1 \le l \le m}[\Sigma_{b \in L_a} C_c((b,a) \rightarrow (p_k, p_l))]$ where l denotes any processor connected to processor k, and define $AP_a = C_p(a \rightarrow p_k)$. If there exists no processor l connected to processor k, then AC_a is set infinite.

Step 5): Let $I_a = \min(AC_a, AP_a)$. Then compute the heuristic function $h(n)$ as

$$h(n) = \sum_{a \in B} I_a.$$

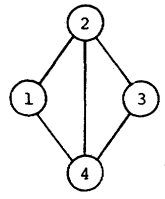
It is not difficult to see that $h(n)$ is a consistent lower bound of $h^*(n)$ for all n, so the state-space search will be guaranteed to find an optimal weak homomorphism using the algorithm described in the following.

Algorithm 2: Find an optimal weak homomorphism between two graphs.

Step 1): Put the initial node $K = \emptyset$ on a list called OPEN, and set $f(K) = 0$ where f is an evaluation function defined by (5).

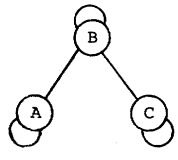
Step 2): Remove from OPEN the node n with a smallest f value and put it on a list called CLOSED.

Step 3): If n satisfies the goal state defined in the previous section, then find out the corresponding graph mapping M as the desired homomorphism and exit. Otherwise, continue.




```
T = (V_T, E_T) where
V_T = {1,2,3,4},
E_T = {(1,2),(2,3),(3,4),(4,1),(2,4)}.
```

Fig. 3. Task graph.




```
P = (V_P, E_P) where
V_P ={ A,B,C},
E_P = {(A,B),(B,C),(A,A),(B,B),(C,C)}.
```

Fig. 4. Processor graph.

Step 4): Expand node n, using operators applicable to n, and compute the value $f(n') = g(n') + h(n')$ for each successor n' of n where $g(n')$ is computed according to (1), (2), (3), and (4), and $h(n')$ is computed according to Algorithm 1. Put all the successors of n on OPEN.

Step 5): Go to Step 2).

In Step 4), if $h(n')$ is always set 0, then Algorithm 2 is a uniform-cost search. Otherwise, it is an A^* algorithm.

V. Illustrative Example

Here, we give an example to illustrate the mapping between the task graph and the processor graph by the proposed state-space search method. Task graph T and processor graph P are shown in Figs. 3 and 4.

The intermodule communication time and the module processing time are listed in Tables I and II.

Using the vertex and edge transformation notations defined previously, we have

$$\begin{aligned} C_p(1 \rightarrow A) &= 10, & C_p(3 \rightarrow A) &= 70, \\ C_p(1 \rightarrow B) &= 20, & C_p(3 \rightarrow B) &= 50, \\ C_p(1 \rightarrow C) &= 30, & C_p(3 \rightarrow C) &= 80, \\ C_p(2 \rightarrow A) &= 40, & C_p(4 \rightarrow A) &= 50, \\ C_p(2 \rightarrow B) &= 5, & C_p(4 \rightarrow B) &= 80, \\ C_p(2 \rightarrow C) &= 10, & C_p(4 \rightarrow C) &= 20, \end{aligned}$$

and

$$\begin{aligned} C_c((1,2) \rightarrow (A,B)) &= C_c((1,2) \rightarrow (B,C)) = 20, \\ C_c((2,3) \rightarrow (A,B)) &= C_c((2,3) \rightarrow (B,C)) = 5, \\ C_c((3,4) \rightarrow (A,B)) &= C_c((3,4) \rightarrow (B,C)) = 35, \\ C_c((4,1) \rightarrow (A,B)) &= C_c((4,1) \rightarrow (B,C)) = 40, \\ C_c((2,4) \rightarrow (A,B)) &= C_c((2,4) \rightarrow (B,C)) = 70. \end{aligned}$$

Then, using Algorithm 2 as an ordered-search algorithm, we can find an optimal weak homomorphism M^*: $V_T \rightarrow V_P$ and compute the minimum task turnaround time. Fig. 5

TABLE I
INTERMODULE COMMUNICATION TIME

		module 1	2	3	4
module	1	0	20	0	40
	2	20	0	5	70
	3	0	5	0	35
	4	40	70	35	0

TABLE II
MODULE PROCESSING TIME

		processor A	B	C
module	1	10	20	30
	2	40	5	10
	3	70	50	80
	4	50	80	20

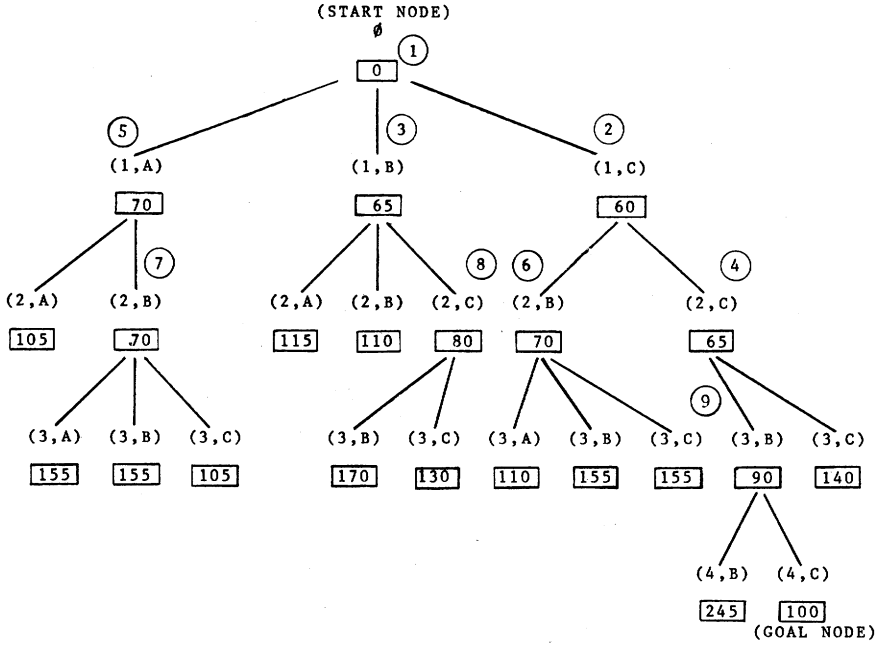


Fig. 5. State-space search tree of the illustrative example (circled numbers indicate node expansion order, and squared numbers indicate node costs).

shows the resulting search tree. The optimal weak homomorphism M^*, as shown, is

$$M^*: \begin{cases} 1 \rightarrow C \\ 2 \rightarrow C \\ 3 \rightarrow B \\ 4 \rightarrow C \end{cases}$$

which defines the optimal assignment of task T to distributed computing system P, and the minimum task turnaround time of this assignment is 100.

As shown in Fig. 5, totally only 9 state-space nodes are expanded and 23 nodes generated before the goal node is found. The search concentrates rather well on the right path to the goal node with only a few branches elsewhere. To prove that such concentration is a result from utilizing the heuristics proposed previously, we further reuse Algorithm 2 as a uniform-cost algorithm (by setting $h(n)$ zero for all nodes) to find the optimal weak homomorphism again. The resulting state-space search tree is too large to be included here as a figure. We only mention that totally 22 state-space nodes are expanded in the tree with 49 nodes generated.

More experiments have been performed on a CDC CYBER 170/720 computer for some other cases to test the effectiveness of the heuristic function $h(n)$ used in Algorithm 2. The results [18] show that in most cases with module numbers and processor numbers less than 10, the number of generated nodes by Algorithm 2 is about a half of that generated by the uniform-cost algorithm. It is also found in these cases that the complexity of Algorithm 2 as defined by the number of generated nodes is approximately to the order of N^2, although

optimal task assignment, like many other graph matching problems, needs exponential time in the worst case [17].

VI. Conclusions

In this paper, we describe a graph matching approach to task assignment optimization. Optimal task assignment is transformed into the search problem of optimal weak graph homomorphism, which is then solved by heuristic state-space search using the A^* algorithm. The approach uses the task turnaround time as the cost measure and the minimax criterion as the criterion for assignment optimization. This approach allows various system constraints to be easily included for consideration. It minimizes interprocessor communication and optimizes load balancing by minimizing the task turnaround time.

There are M^N possible assignments to be considered in an assignment problem with M processors and N modules, and actual computation using an enumerative algorithm requires time to the order of M^N [10]. The state-space heuristic search algorithm proposed in this paper is shown effective by an illustrative example and some case studies for speeding up solution search. Further research can be directed to the improvement of the proposed lower-bound estimate $h(n)$ of $h^*(n)$ in Algorithm 1 and the inclusion of precedence consideration into the proposed algorithms.

References

[1] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *Computer,* vol. 15, pp. 50–56, June 1982.

[2] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *Computer,* vol. 13, pp. 57–69, Nov. 1980.

[3] H. S. Stone and S. H. Bokhari, "Control of distributed processes," *Computer,* vol. 11, pp. 97–106, July 1978.

[4] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Eng.,* vol. SE-3, pp. 85–93, Jan. 1977.

[5] T. C. K. Chow and J. A. Abraham, "Load balancing in distributed systems," *IEEE Trans. Software Eng.,* vol. SE-8, July 1982.

[6] S. H. Bokhari, "Dual processor scheduling with dynamic reassignment," *IEEE Trans. Software Eng.,* vol. SE-5, July 1979.

[7] ——, "On the mapping problem," *IEEE Trans. Comput.,* vol. C-30, pp. 207–214, Mar. 1981.

[8] H. S. Stone, "Critical load factors in two-processor distributed systems," *IEEE Trans. Software Eng.,* vol. SE-4, pp. 254–258, May 1978.

[9] G. S. Rao, H. S. Stone, and T. C. Hu, "Assignment of tasks in a distributed processor system with limited memory," *IEEE Trans. Comput.,* vol. C-28, pp. 291–299, Apr. 1979.

[10] C. C. Price, "The assignment of computational tasks among processors in a distributed system," in *Proc. Nat. Comput. Conf.,* May 1981, pp. 291–296.

[11] L. Kleinrock and A. Nilsson, "On optimal scheduling algorithms for time-shared systems," *J. ACM,* vol. 28, no. 3, pp. 477–486, July 1981.

[12] Y. C. Chow and W. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.,* vol. C-28, pp. 354–361, May 1979.

[13] W. H. Tsai, "Graph matching problems: A survey and a tutorial," in *Proc. 1st Conf. Comput. Algorith.,* Hsinchu, Taiwan, China, July 1982.

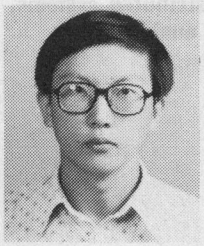
[14] W. H. Tsai and K. S. Fu, "Subgraph error-correcting isomorphisms for syntactic pattern recognition," *IEEE Trans. Syst. Man, Cybern.,* vol. SMC-13, pp. 48–62, Jan./Feb. 1983.

[15] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.

[16] P. R. Ma, E. Y. S. Lee, and M. Tsuchiya, "A task allocation model for distributed computing systems," *IEEE Trans. Comput.,* vol. C-31, pp. 41–47, Jan. 1982.

[17] R. C. Read and D. G. Corneil, "The graph isomorphism disease," *J. Graph Theory,* pp. 339–363, 1977.

[18] C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," Inst. Comput. Eng., Nat. Chiao Tung Univ., Hsinchu, Taiwan, Republic of China, Tech. Rep., July 1983.



Chien-Chung Shen was born in Taiwan, Republic of China, on July 15, 1960. He received the B.S. and M.S. degrees in computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982 and 1984, respectively.

From 1982 to 1984, he served as a Teaching and Research Assistant in the Institute of Computer Engineering, National Chiao Tung University. His research interests include artificial intelligence, pattern recognition, graph theory, database design, and computer-aided design of VLSI.

Wen-Hsiang Tsai (S'79–M'80) was born in Tainan, Taiwan, Republic of China, on May 10, 1951. He received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1973, the M.S. degree from Brown University, Providence, RI, in 1977, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1979, all in electrical engineering.

From 1973 to 1975, he served in the Chinese Navy as an Electronics Officer. From 1977 to 1979, he worked as a Research Assistant in the Advanced Automation Research Laboratory, School of Electrical Engineering, Purdue University. Since November 1979, he has been on the faculty of the Institute of Computer Engineering at National Chiao Tung University, Hsinchu, Taiwan. His current research interests are image processing and pattern recognition, computer vision applications in robotics and automation, and parallel processing and multiprocessor systems.