On the Number of Acceptable Task Assignments in Distributed Computing Systems

KANG G. SHIN, SENIOR MEMBER, IEEE, AND MING-SYAN CHEN

Abstract—**In a distributed computing system, a job is usually decomposed into several cooperating tasks which are then assigned to a set of processors in the system to exploit the inherent parallelism in job execution. The distributed computing system and cooperating tasks can be represented by a *processor graph* $G_P = (V_P, E_P)$ and a *task graph* $G_T = (V_T, E_T)$, respectively. An edge between a pair of nodes in G_T represents the existence of direct communications between the two corresponding tasks. The maximal number of hops between two processors in G_P to which two adjacent tasks in G_T are assigned is called *dilation* of that assignment. For obvious reasons, it is important to keep the communication delay between any two adjacent tasks in G_T low. This can be accomplished by keeping the dilation of an assignment below some specified value. An assignment is said to be *acceptable* if its dilation is less than or equal to the specified value.**

Characterization and use of the number of acceptable assignments for given G_T and G_P are the subject of this paper. First, assignments with the dilation less than or equal to one are considered. This dilation constraint represents a special case in which two adjacent tasks in G_T must be assigned to either a single processor or two adjacent processors in G_P. Let $N(G_T, G_P)$ denote the number of acceptable assignments under this constraint. We not only derive bounds of $N(G_T, G_P)$ for arbitrary G_T and G_P, but also formulate a recursive expression for $N(G_T, G_P)$ when G_T is a tree. For some restricted cases, either closed-form or recursive-form expressions of $N(G_T, G_P)$ are derived. The knowledge of $N(G_T, G_P)$ is shown to be useful not only in designing a processor interconnection structure but also in analyzing as well as improving the state-space search for the task assignment problem. Finally, we extend our results on $N(G_T, G_P)$ to the completely general case—assignments with dilations greater than one—where two adjacent tasks in G_T can be assigned to *any* two processors in G_P which are not necessarily adjacent to each other.

Index Terms—**Acceptable task assignment, adjacency requirement, dilation, processor and task graphs, state-space search.**

I. INTRODUCTION

THE availability of inexpensive, high-performance microprocessors and memory chips has made it attractive to build distributed computing systems with these components. In such a system, a job is usually decomposed into a set of cooperating tasks which are then assigned to a set of processors in order to exploit the inherent parallelism in job execution [1]–[4]. Each job can thus be described by an indirected graph called the *task graph*, $G_T = (V_T, E_T)$, where V_T is the set of nodes (vertices), each representing a task of the job, and $E_T \subseteq V_T \times V_T$ is the set of edges, each representing intertask communications between the two task nodes connected by the edge. When there is an edge between two task nodes in G_T, the two tasks are said to be *related* to each other. Similarly, a distributed computing system can be represented by an undirected graph called the *processor graph*, $G_P = (V_P, E_P)$, where V_P is the set of processors in the system and $E_P \subseteq V_P \times V_P$ is the set of edges representing communication links between processors.

The maximal number of hops between two processors to which two related tasks are assigned is called the *dilation* of that assignment [5]. Cooperating tasks of a job are usually required to be assigned to a set of processors in such a way that the communication delay between any two related tasks must be kept low. One way to accomplish this is to limit the dilation to a small number so that two related tasks may be assigned to those processors located physically close to each other. An assignment is said to be *acceptable* if its dilation is less than or equal to prespecified integer value.

The problem of deriving an "optimal" (in the sense of, for example, load balancing or minimization of job execution time) task assignment is very hard and known to be NP-complete [6], [7]. In [4], the task assignment problem is formulated as a state-space search problem which is then solved by the A^* algorithm [8]. However, without the knowledge of the number of acceptable assignments for given G_T and G_P, one cannot tell the size of the state-space to be searched in the A^* algorithm. Note that such an algorithm often requires a large number of evaluations of a complex heuristic function.

As will be discussed later, the knowledge of the number of acceptable assignments can be used not only for providing a simplified state-space search but also for reducing the size of the state-space to be searched. Although a search method using this knowledge may reach a suboptimal goal node instead of the optimal one, it requires much less computation cost and, thus, provides a useful insight into the state-space search. In addition, the knowledge of the number of acceptable assignments and its relation with the processor and task graphs can play an important role in the design of a distributed computing system. In other words, one can derive the system's structure from this knowledge by maximizing the number of acceptable assignments for a given set of cooperating tasks. For the rea-

Manuscript received June 4, 1987; revised October 14, 1987 and January 28, 1988. This work was supported in part by the Office of Naval Research under Contract N00014-85-K-0122. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not reflect the view of ONR.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8931163.

sons mentioned above, we shall concentrate on obtaining the number of acceptable task assignments for given G_T and G_P.

Notice that the necessity of fast communication between two related tasks has usually made it important to assign them to either a single processor or two adjacent processors, i.e., the dilation of an assignment is kept less than or equal to one. Let $N(G_T, G_P)$ denote the number of acceptable assignments under this constraint. As it will be pointed out later, our results on $N(G_T, G_P)$ can be extended and applied to the completely general case, i.e., those assignments with the dilation greater than one. Thus, without loss of generality, we shall henceforth address only the formulation and application of $N(G_T, G_P)$. Unless mentioned otherwise, in what follows, an acceptable assignment is referred to as an assignment with the dilation less than or equal to one.

To facilitate our discussion, the task assignment problem can be transformed and stated formally as follows. Given the task graph G_T and processor graph G_P, we want to label nodes in G_T with the nodes in G_P in such a way that each node in G_T is labeled with exactly one node from G_P and every pair of adjacent nodes in G_T is labeled with either a single node or two adjacent nodes in G_P. This constraint will be termed *adjacency requirement* and every labeling satisfying the adjacency requirement is called an *acceptable labeling*. The actual task assignment is to choose one from the set of acceptable labelings that minimizes/maximizes the associated criterion function. Note that we are addressing the problem of determining the number of acceptable assignments, rather than the determination of task assignments themselves. This fact distinguishes our work from other related works [2], [4], [9]–[11].

The paper is organized as follows. Section II provides a brief introduction of necessary notation and definitions. Our main results are given in Sections III and IV. Section III deals with the derivation of $N(G_T, G_P)$. First, we derive the bounds of $N(G_T, G_P)$, when G_P and G_T are arbitrary. Then, some important special cases are treated: 1) when G_T is a tree and G_P is arbitrary, and 2) when G_T is a tree and G_P is an n-regular graph. For case 1) we shall develop a recursive formula to obtain the exact number of $N(G_T, G_P)$. 2) is a special case of 1) for which the exact number of acceptable assignments can be determined in a closed form. Also, we shall determine expressions for the case when G_P or G_T is restricted to a certain family of graphs. Three application examples are presented in Section IV to illustrate how the knowledge of the number of acceptable assignments can be used. In light of these examples, some remarks on the task assignment problem are also made. More importantly, it is shown that our results are extensible to the completely general case, i.e., those assignments with dilations greater than one. Finally, the paper concludes with Section V.

II. Notation and Definitions

A graph G_α is said to be a *spanning subgraph* of another graph G_β if $V_\alpha = V_\beta$ and $E_\alpha \subseteq E_\beta$ [12]. A *complete subgraph* of a graph G is a subgraph which is a complete graph,[1] and a *component* is a maximal complete subgraph which is not a proper subgraph of any other complete subgraph of G. Thus, any graph can be viewed as the union of all of its components. Let H_i, $1 \leq i \leq r$, denote the components in a graph G, where r is the number of components in G. The *redundance* sets of G are defined as $B_i = H_i \cap \{\bigcup_{j=1}^{i-1} H_j\}$ for $1 \leq i \leq r$. For example, the graph in Fig. 1(a) is the union of its components in Fig. 1(b), where $H_1 = \{1, 2, 3\}$, $H_2 = \{2, 3, 4\}$, and $H_3 = \{4, 5\}$. The redundance sets of this graph are $B_1 = \emptyset$, $B_2 = \{2, 3\}$, and $B_3 = \{4\}$.

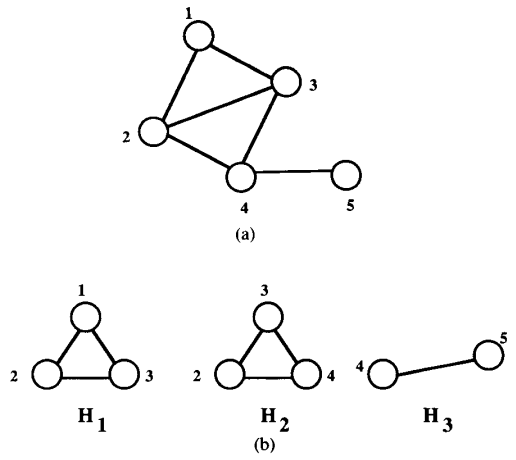

Fig. 1. Decomposition of a graph into its components.

The symbol U_m is used to denote an m-dimensional vector of which all entries are one, and P_n, C_n, S_n, and K_n to denote, respectively, a path, cycle, star, and complete graph with n nodes [12]. Examples for P_4, C_4, S_4, and K_4 are given in Fig. 2. Also, Q_n is used to denote an n-dimensional cube [12], and Q_3 is shown in Fig. 2(e) as an example. Besides, R_n denotes an n-regular graph in which every node has the same degree n. Unless explicitly specified otherwise, every vector referred to in this paper is treated as a column vector of positive integers, and all graphs are assumed to be connected. In addition, the following definitions are necessary to proceed with our discussion.

Definition 1: The *adjacency matrix* $\boldsymbol{M} = [M_{ij}]$, of a labeled graph G with m nodes is an $m \times m$ matrix in which $M_{ij} = 1$ if node i is adjacent to node j in G or $i = j$, and $M_{ij} = 0$ otherwise. For convenience, in the rest of the paper we shall use an $m \times m$ nonnegative symmetric matrix $\boldsymbol{A} = [A_{ij}]$ to denote the adjacency matrix of a processor graph G_P, where $m = |V_P|$.

Definition 2: Among all acceptable labelings of G_T with the nodes of G_P, let $D_i(k)$ represent the number of acceptable labelings in which the node $n_i \in V_T$ is labeled with the value k, i.e., assigned to processor k in V_P. Then, for each node $n_i \in V_T$, the vector $\boldsymbol{D}_i = [D_i(1), D_i(2), \cdots, D_i(m)]^T$ is called the *distribution vector* of n_i, where T denotes "transpose."

Definition 3: The *attaching function* associated with the adjacency matrix $\boldsymbol{A}$, $f_A: \boldsymbol{I}^m \rightarrow \boldsymbol{I}^m$, is defined as $f_A(V) = \boldsymbol{A}V$, $\forall V \in \boldsymbol{I}^m$ where $\boldsymbol{I}^m$ is the set of all m-dimensional vectors of nonnegative integers.

[1] A graph G is said to be *complete* if each node in G is connected to all the other nodes in G.

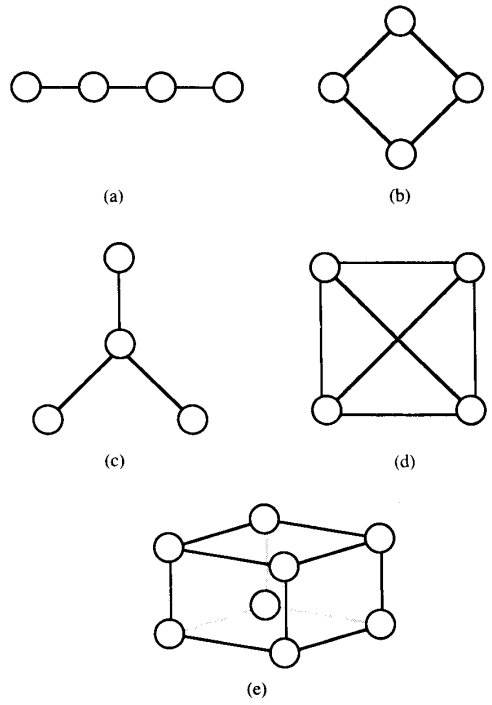

Fig. 2. Various example graphs. (a) P_4. (b) C_4. (c) S_4. (d) K_4. (e) Q_3.

It can be verified that if $\boldsymbol{D}_i$ is the distribution vector of a task node n_i in G_T and if G_T^* is the resulting graph by attaching a new node y to n_i, then $f_A(\boldsymbol{D}_i)$ is the distribution vector of y in G_T^*. For the example processor graph in Fig. 3(a), we get $\boldsymbol{D}_1 = [4, 3, 3, 2]^T$, which means there are four ways to label n_1 with processor 1, three ways to label n_1 with processor 2, and so on. After n_3 is attached to n_1, we have $\boldsymbol{D}_3 = f_A([4, 3, 3, 2]^T) = [12, 10, 10, 6]^T$ in Fig. 3(c). Notice that, to satisfy the adjacency requirement, n_3 in G_T^* can be labeled with processor 2 only when n_1 in G_T was labeled with any of processor 1, processor 2, and processor 3. Then, we have $D_3(2) = 4 + 3 + 3 = 10$. The other entries in $\boldsymbol{D}_3$ can be obtained similarly.

Also, introduced are the following four definitions, which will be very useful in determining $N(G_T, G_P)$ when G_T is a tree.

Definition 4: The *product* of two vectors, denoted by $\odot$, is defined as $\boldsymbol{V}_3 = \boldsymbol{V}_1 \odot \boldsymbol{V}_2$ iff $V_3(k) = V_1(k)V_2(k)$ for $1 \le k \le m$, $\forall \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3 \in \boldsymbol{I}^m$, where $V_i(k)$ denotes the kth element of the vector $\boldsymbol{V}_i$ for $i = 1, 2, 3$. Clearly, the product of vectors is associative and commutative. We shall use $\Pi_{i=1}^{q} \boldsymbol{V}_i$ to denote the product of q vectors. Note that this operation is not the same as the conventional inner product of vectors, since it results in a vector, rather than a scalar.

Definition 5: The *multiplication* of two vectors associated with the adjacency matrix $\boldsymbol{A}$, denoted by $*_A$, is defined as

$$\boldsymbol{V}_1 *_A \boldsymbol{V}_2 = \boldsymbol{V}_1 \odot f_A(\boldsymbol{V}_2) \quad \forall \boldsymbol{V}_1, \boldsymbol{V}_2 \in \boldsymbol{I}^m.$$

Definition 6: The *sorted vector* of a vector $\boldsymbol{V} \in \boldsymbol{I}^m$, denoted by $\boldsymbol{V}^* \in \boldsymbol{I}^m$, is a vector whose components $V^*(i)$, $1 \le i \le m$, are a descending permutation of the components of $\boldsymbol{V}$.

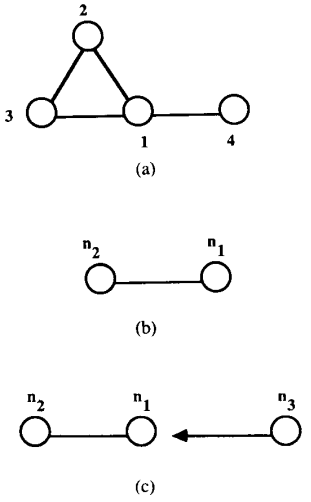

Fig. 3. An illustrative example for the attaching function. (a) G_P. (b) G_T. (c) G_T.

Definition 7: The *weight of a vector*, $W: \boldsymbol{I}^m \to \boldsymbol{I}$, is defined as

$$W(\boldsymbol{V}) = \sum_{i=1}^{m} V(i), \quad \forall \boldsymbol{V} \in \boldsymbol{I}^m.$$

III. Derivation of $N(G_T, G_P)$

A. The Case of Arbitrary Processor and Task Graphs

The following lemma immediately follows from the adjacency requirement in task assignment.

Lemma 1:

a) $N(G_T, G_P) \le N(G_T, G_P')$ if G_P is a spanning subgraph of G_P'.

b) $N(G_T, G_P) \ge N(G_T', G_P)$ if G_T is a spanning subgraph of G_T'.

By a) of this lemma, the inequality $N(G_T, G_P) \le N(S(G_T), G_P)$ always holds, where $S(G_T)$ is an arbitrary spanning tree of G_T. Moreover, we have the following theorem.

Theorem 1: For any arbitrary G_P and G_T, there exist the following bounds of $N(G_T, G_P)$, which are independent of $|E_T|$:

a) $N(G_T, G_P) \ge N(K_{|V_T|}, G_P) = \sum_{i=1}^{r} (|H_i|^{|V_T|} - |B_i|^{|V_T|})$

b) $N(G_T, G_P) \le N(S(G_T), G_P) \le N(S_{|V_T|}, G_P) = \sum_{i=1}^{|V_P|} (d_i + 1)^{|V_T|-1}$, where, as before, H_i and B_i, $1 \le i \le r$, are respectively the component and redundance sets of G_P, d_i is the degree of node i in G_P, and $S_{|V_T|}$ is a star with $|V_T|$ nodes.

Proof: a) Since the complete graph $K_{|V_T|}$ possesses the maximal number of edges among all the graphs with $|V_T|$ nodes, the inequality $N(G_T, G_P) \ge N(K_{|V_T|}, G_P)$ follows from b) of Lemma 1. Every node in $K_{|V_T|}$ is adjacent to all the other nodes; to satisfy the adjacency requirement, all

the nodes in $K_{|V_T|}$ must be labeled with nodes within one component in G_P. There are $|H_i|^{|V_T|}$ ways to label G_T with nodes in the component set H_i of G_P, and, thus, the total number of such labelings can be obtained by adding up all $|H_i|^{|V_T|}, 1 \leq i \leq r$, where r is the number of components of G_P. However, the number of labelings with a component set H_i contains double counts for those labelings in which all the nodes in $K_{|V_T|}$ are labeled with nodes in the redundant set B_i. The redundant counts must be removed by subtracting $|B_i|^{|V_T|}$ from $|H_i|^{|V_T|}$, and thus, a) follows.

Because the proof of b) requires more bases to cover, we shall complete the proof of b) after Theorem 2. Q.E.D.

Notice that when G_P is a complete graph, the number of acceptable assignments is $|V_P|^{|V_T|}$ regardless of the type of G_T. This fact implies that the tightness of the bounds depends strongly on the structure of G_P. From Theorem 1, we have the following corollary for the lower bound of $N(G_T, G_P)$ when G_P is restricted to a hypercube or a cycle. Recall that Q_n denotes an n-dimensional cube and C_m is a cycle with m nodes.

Corollary 1.1:

a) $N(K_h, Q_n) = 2^{n+h-1}n - 2^n(n-1)$,

b) $N(K_h, C_m) = 2^h m - m$.

Proof: Since there is no K_3 subgraph in Q_n, all the nodes in K_h must be labeled with either a single node or two adjacent nodes in Q_n. There are 2^h ways to label K_h with each pair of adjacent nodes in Q_n, and the number of edges in Q_n is $2^{n-1}n$. Therefore, when each edge in Q_n is considered separately, the total number of acceptable labelings is $2^{n-1}n2^h$. However, the labeling in which all the nodes in K_h are labeled with the same node from Q_n occurs n times. Thus, we have to remove the redundant counts by subtracting $2^n(n-1)$ from $2^{n-1}n2^h$, leading to $N(K_h, Q_n) = 2^{n+h-1}n - 2^n(n-1)$.

It can be easily seen that b) is valid when $m = 3$. Note that there is no K_3 subgraph in C_m, $\forall m > 3$. Thus, b) can be proved similarly. Q.E.D.

As shown above, one can derive only bounds[2] of $N(G_T, G_P)$ when G_T and G_P are both arbitrary graphs. However, when G_T is restricted to a tree, $N(G_T, G_P)$ can be expressed in a recursive form as will be shown in the following subsection. Moreover, it will be shown in Section III-C that $N(G_T, G_P)$ can be expressed in a closed form when G_T is a tree and G_P is an n-regular graph R_n.

B. The Case when G_T is a Tree

To derive the recursive formula for $N(G_T, G_P)$ when G_T is a tree and G_P is arbitrary, we must convert G_T to a rooted tree by choosing an arbitrary node of G_T as the root. Let us define the *carrying vector* of a node of G_T as follows.

Definition 8: The *carrying vector* of a node $n_i \in V_T$, denoted by $\boldsymbol{Y}_i$, is defined in a recursive form,

$$\boldsymbol{Y}_i = \begin{cases} \boldsymbol{U}_m & \text{if } n_1 \text{ is a leaf,} \\ \prod\limits_{n_j \in C(n_i)} f_A(\boldsymbol{Y}_j) & \text{otherwise} \end{cases} \quad (1)$$

where $C(n_i)$ represents the set of children of the node n_i.

[2] These bounds are necessarily loose because of the wide range of structural variations in the processor and task graphs.

As we shall prove later in Theorem 2, the carrying vector is so defined that we can determine the distribution vector of any node, say n_k, just by rearranging the tree to make n_k the root and then computing the carrying vector of the node n_k by (1).

Let n_1 and n_2 be two nodes with distribution vectors $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ in task graphs G_1 and G_2, respectively. Suppose G' is the resulting graph by adding a new edge between n_1 and n_2 to connect G_1 and G_2. Then, the distribution vectors of the two nodes n_1 and n_2 in G' can be determined by the following lemma.

Lemma 2: Let $\boldsymbol{D}'_1$ and $\boldsymbol{D}'_2$ denote, respectively, the resulting distribution vectors of task nodes n_1 and n_2 after connecting n_1 and n_2 with a new edge. Then

a) $\boldsymbol{D}'_1 = \boldsymbol{D}_1 *_A \boldsymbol{D}_2$

b) $\boldsymbol{D}'_2 = \boldsymbol{D}_2 *_A \boldsymbol{D}_1$ where $\boldsymbol{A}$ is the adjacency matrix of G_P.

Proof: Consider part a). When n_1 in G_1 and n_2 in G_2 are labeled, respectively, with nodes i and j in G_P, $A_{ij} = A_{ji} = 1$ is the necessary and sufficient condition that this labeling is still acceptable for the resulting graph G'. The number of labelings of G_2 which are still acceptable after connecting n_1 and n_2 is $\sum_{j=1}^{m} A_{ji}D_2(j)$. Since the number of different acceptable labelings of G_1 in which n_1 is labeled with i is $D_1(i)$, we get $D'_1(i) = D_1(i) \sum_{j=1}^{m} A_{ji}D_2(j)$ and part a) follows. Part b) can be proved similarly. Q.E.D.

Thus, we have the following important result.

Theorem 2: The carrying vector of the root node of a tree is the same as its distribution vector, i.e., $\boldsymbol{D}_r = \boldsymbol{Y}_r$ if n_r is the root of a tree.

Proof: We prove this theorem by induction. Obviously, the theorem holds for a trivial tree (i.e., a tree with only one node). In that case, both the distribution and carrying vectors are $\boldsymbol{U}_m$, where $m = |V_P|$ as before.

Assume that the theorem holds for all the children of a node, say n_r. Let n_{r_i}, $1 \leq i \leq c$, denote the node n_r's children, i.e., $C(n_r) = \{n_{r_i} | 1 \leq i \leq c\}$, where $c = |C(n_r)|$. Then, from Lemma 2 the distribution vector of n_r with only one child n_{r_1} is $\boldsymbol{U}_m *_A \boldsymbol{Y}_{r_1}$. Thus, by attaching one more child at a time, the distribution vector of n_r with all its children attached becomes

$$\begin{aligned} \boldsymbol{D}_r &= \boldsymbol{U}_m *_A \boldsymbol{Y}_{r_1} *_A \boldsymbol{Y}_{r_2} *_A \cdots *_A \boldsymbol{Y}_{r_c} \\ &= [\boldsymbol{U}_m \odot (\boldsymbol{A}\boldsymbol{Y}_{r_1})] \odot (\boldsymbol{A}\boldsymbol{Y}_{r_2}) \odot \cdots \odot (\boldsymbol{A}\boldsymbol{Y}_{r_c}) \\ &= \prod_{i=1}^{c} \boldsymbol{A}\boldsymbol{Y}_{r_i} \\ &= \prod_{n_j \in C(n_r)} f_A(\boldsymbol{Y}_j) \\ &= \boldsymbol{Y}_r. \end{aligned}$$ Q.E.D.

Given an arbitrary G_P, one can compute the carrying vector of all nodes in a rooted tree G_T by applying (1) recursively, and determine the number of acceptable labelings from the corollary below.

Corollary 2.1: $\forall n_i \in V_T$, $W(\boldsymbol{Y}_i) = N(T(n_i), G_P)$, where $T(n_i)$ is the tree formed by the node n_i and its descendants.

Proof: Suppose $T(n_i)$ is the task tree, then we have $\boldsymbol{D}_i = \boldsymbol{Y}_i$ from Theorem 2. This corollary follows from the fact that $W(\boldsymbol{D}_i) = N(T(n_i), G_P)$. Q.E.D.

The computational complexity in obtaining $N(G_T, G_P)$, when G_T is a tree, can be determined as follows. By Lemma

2 and Theorem 2, $N(G_T, G_P)$ can be obtained by calculating the carrying vector of each node in the tree with (1). The complexity in obtaining $f_A(Y_j)$ from given A and Y_j is $O(m^2)$, where $m = |V_P|$, $Y_j \in I^m$ and A is an $m \times m$ matrix. Note that the complexity of the operation $\odot$ is $O(m)$. Thus, when G_T is a tree, the complexity in calculating $N(G_T, G_P)$ is $O(|V_T| - 1)O(m^2 + m) = O(|V_T|m^2)$.

To illustrate the ideas presented thus far, consider the example processor and task graphs shown in Fig. 4(a) and (b). We have the following adjacency matrix for G_P in Fig. 4(a).

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Suppose n_3 in G_T is chosen as the root of the tree. Then, we get the rooted tree in Fig. 4(c) where n_1, n_5, and n_6 are leaf nodes (i.e., $Y_1 = Y_5 = Y_6 = [1 \quad 1 \quad 1 \quad 1]^T$), and $C(n_3) = \{n_2, n_4\}$, $C(n_2) = \{n_1\}$ and $C(n_4) = \{n_5, n_6\}$. Thus, we get

$$Y_2 = f_A(Y_1) = AY_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 3 \end{bmatrix}$$

$$Y_4 = (AY_5) \odot (AY_6) = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 3 \end{bmatrix} \odot \begin{bmatrix} 2 \\ 4 \\ 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 16 \\ 9 \\ 9 \end{bmatrix}$$

$$Y_3 = (AY_2) \odot (AY_4) = \begin{bmatrix} 6 \\ 12 \\ 10 \\ 10 \end{bmatrix} \odot \begin{bmatrix} 20 \\ 38 \\ 34 \\ 34 \end{bmatrix} = \begin{bmatrix} 120 \\ 456 \\ 340 \\ 340 \end{bmatrix}$$

$$W(Y_3) = 120 + 456 + 340 + 340 = 1256.$$

From Fig. 4(a) we get $H_1 = \{2, 3, 4\}$, $H_2 = \{1, 2\}$, and then $B_1 = \emptyset$ and $B_2 = \{2\}$. It can be verified that $3^6 + 2^6 - 1^6 = 792 < 1256 < 2^5 + 4^5 + 3^5 + 3^5 = 1542$, which agrees with the bounds in Theorem 1.

In what follows, we shall prove part b) of Theorem 1. To facilitate the proof, we need the following definition [13]. Also, recall that V^* denotes the sorted vector of V.

Definition 9: Let V_1 and V_2 be two m-dimensional vectors. V_1 is said to be *weakly submajorized* by V_2, denoted by $V_1 <_w V_2$, if $\sum_{i=1}^{k} V_1(i) \le \sum_{i=1}^{k} V_2(i)$ for $1 \le k \le m$.

Then, we have the following three propositions.

Proposition 1: If $V_1 <_w V_2$, then $V_1^* \odot V_3^* <_w V_2^* \odot V_3^*$, $\forall V_1, V_2, V_3 \in I^m$.

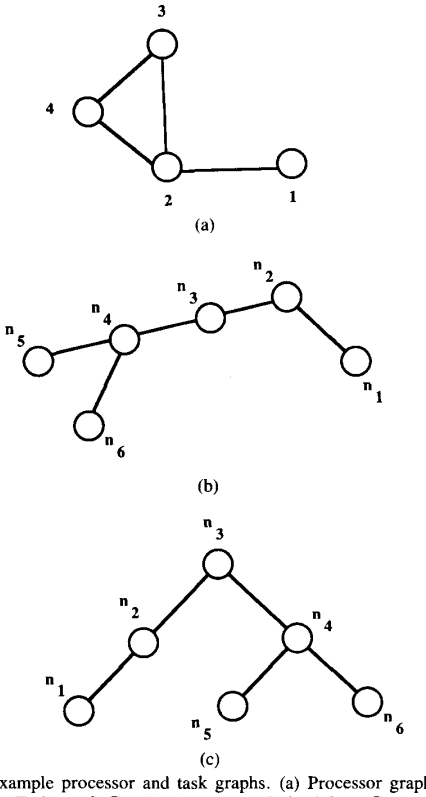

Fig. 4. Example processor and task graphs. (a) Processor graph G_P. (b) Task graph G_T. (c) A root tree derived from G_T.

Proof: Let $V_1^* = [a_1, a_2, \cdots, a_m]^T$, $V_2^* = [b_1, b_2, \cdots, b_m]^T$, $V_3^* = [c_1, c_2, \cdots, c_m]^T$, and $\delta_i = V_1^*(i) - V_2^*(i) = a_i - b_i$. Since $V_1 <_w V_2$, $\sum_{i=1}^{k} \delta_i \le 0$ for $1 \le k \le m$. We obtain

$$\begin{aligned} \sum_{i=1}^{k} c_i a_i - \sum_{i=1}^{k} c_i b_i &= \sum_{i=1}^{k} c_i (a_i - b_i) \\ &= c_1\delta_1 + c_2\delta_2 + \sum_{i=3}^{k} c_i\delta_i \\ &\le c_2(\delta_1 + \delta_2) + \sum_{i=3}^{k} c_i\delta_i \\ &\quad \text{(Since } \delta_1 \le 0 \text{ and } \{c_i\} \text{ is decreasing.)} \\ &\le c_u \sum_{i=1}^{u} \delta_i + \sum_{i=u+1}^{k} c_i\delta_i \\ &\le c_k \sum_{i=1}^{k} \delta_i \le 0, \end{aligned}$$

and this proposition follows. Q.E.D.

Proposition 2: Given a vector $V \in I^m$ and a decreasing sequence of node degrees, $\{d_i\}_{i=1}^m$, in a graph with an adjacency matrix A, if $V <_w [(d_1+1)^n, (d_2+1)^n, \cdots, (d_m+1)^n]^T$, then $f_A(V) <_w [(d_1+1)^{n+1}, (d_2+1)^{n+1}, \cdots, (d_m+1)^{n+1}]^T$.

Proof: Let $Z = f_A(V)$. Note that $W(Z) = W(Z^*)$

is the summation of all the elements in the set $L = \{V(1), \cdots, V(1), V(2), , \cdots, V(2), \cdots, V(m), \cdots, V(m)\}$ in which $V(i)$ appears $d_i + 1$ times. Since $\sum_{i=1}^{k} \boldsymbol{Z}^*(i)$ is the summation of no more than $\sum_{i=1}^{k} (d_i + 1)$ elements from L, we obtain $\sum_{i=1}^{k} \boldsymbol{Z}^*(i) \leq \sum_{i=1}^{k} \boldsymbol{V}^*(i)(d_i + 1) \leq \sum_{i=1}^{k} (d_i + 1)^{n+1}$ (by Proposition 1). Thus, the proposition follows. Q.E.D.

Proposition 3: Let $\{p_i\}_{i=1}^{m}$ be a decreasing sequence of positive integers, and $\boldsymbol{V}_1$, $\boldsymbol{V}_2$, and $\boldsymbol{V}_3$ be m-dimensional vectors such that for some positive integers r and s, $\boldsymbol{V}_1 <_w [p_1^r, p_2^r, \cdots, p_m^r]^T$ and $\boldsymbol{V}_2 <_w [p_1^s, p_2^s, \cdots, p_m^s]^T$. Then, $\boldsymbol{V}_3 = \boldsymbol{V}_1 \odot \boldsymbol{V}_2 <_w [p_1^{r+s}, p_2^{r+s}, \cdots, p_m^{r+s}]^T$.

Proof: For $1 \leq k \leq m$,

$$\sum_{i=1}^{k} V_3^*(i) \leq \sum_{i=1}^{k} V_1^*(i) V_2^*(i)$$

$$\leq \sum_{i=1}^{k} V_1^*(i) p_i^s \quad \text{(by Proposition 1)}$$

$$\leq \sum_{i=1}^{k} p_i^{r+s} \quad \text{(by Proposition 1).} \quad \text{Q.E.D.}$$

Without loss of generality, let $\{d_i\}_{i=1}^{m}$ be a decreasing sequence of node degrees in G_P. Then, part b) of Theorem 1 can be proved as follows.

Proof for part b) of Theorem 1: The first inequality of b) in Theorem 1 follows directly from Lemma 1. To obtain the equality, $N(S_{|V_T|}, G_P) = \sum_{i=1}^{|V_T|} (d_i + 1)^{|V_T|-1}$, consider the case where one more node is to be attached to the central node of a star at a time. Then, this equality follows immediately. Next, we want to prove the second inequality.

We claim that the carrying vector of the root node of a tree with n nodes is weakly submajorized by $[(d_1 + 1)^{n-1}, (d_2 + 1)^{n-1}, \cdots, (d_m + 1)^{n-1}]^T$ and, then, the required result follows from Theorem 2. We prove this claim by induction. Clearly, for a trivial tree, $\sum_{i=1}^{k} \boldsymbol{U}_m(i) \leq \sum_{i=1}^{k} 1, 1 \leq k \leq m$. Next, let s_j be the number of nodes in the tree $T(n_j)$ which is formed by n_j and its descendants. Assume that $\boldsymbol{Y}_j <_w [(d_1 + 1)^{s_j - 1}, (d_2 + 1)^{s_j - 1}, \cdots, (d_m + 1)^{s_j - 1}]^T$. Then, by Proposition 2 we have $f_A(\boldsymbol{Y}_j) <_w [(d_1 + 1)^{s_j}, (d_2 + 1)^{s_j}, \cdots, (d_m + 1)^{s_j}]^T$. Since $\boldsymbol{Y}_i = \Pi_{n_j \in C(n_i)} f_A(\boldsymbol{Y}_j)$, by Proposition 3 we obtain $\boldsymbol{Y}_i <_w [(d_1 + 1)^{s_i - 1}, (d_2 + 1)^{s_i - 1}, \cdots, (d_m + 1)^{s_i - 1}]^T$, where $C(n_i)$ is the set of children of the node n_i and $s_i = \sum_{n_j \in C(n_i)} s_j + 1$. The claim is thus proved by induction and the inequality $N(G_T, G_P) \leq \sum_{i=1}^{|V_T|} (d_i + 1)^{|V_T|-1}$ follows. Q.E.D.

C. Some Restricted Cases

In a more restricted case when G_T is a tree and $G_P = R_n$, $N(G_T, G_P)$ can be expressed in a closed form as given in the following corollary.

Corollary 2.2: If $G_P = R_n$ and G_T is an arbitrary tree, then

$$N(G_T, R_n) = |V_P|(n + 1)^{|V_T|-1}. \tag{3}$$

Proof: If $\boldsymbol{A}$ is the adjacency matrix of an n-regular graph, we have $W(f_A(\boldsymbol{V})) = (n + 1)W(\boldsymbol{V})$, $\forall \boldsymbol{V} \in \boldsymbol{I}^m$. If n_s is an isolated node, then $W(\boldsymbol{D}_s) = W(\boldsymbol{U}_m) = |V_P|$. We can construct any tree by starting with a single node and attaching one node at a time, thereby leading to (3). Q.E.D.

Due to the diversity of network structures, the problem of determining $N(G_T, G_P)$, however, becomes very complicated when G_T is neither a tree nor a complete graph. Although one can derive recursive or closed-form expressions for $N(G_T, G_P)$ when G_T and G_P are restricted to some family of graphs, it is still extremely difficult to determine the general formula of $N(G_T, G_P)$. The procedure of determining $N(C_h, Q_n)$ is presented below to demonstrate the associated difficulty.

To determine the expression of $N(C_h, Q_n)$, consider an alternative approach to the determination of $N(P_h, Q_n)$ without applying Corollary 2.2. Suppose the two end nodes of a (task) path P_h are assigned to processors p_1 and p_2 in Q_n, the distance between which is k. Consider the case where a new task node is to be attached to the end node labeled with p_2. Clearly, the new task node can be assigned to $n + 1$ possible processor nodes in Q_n. From the adjacency requirement and the structure of Q_n, we know that k of these processors have a distance $k - 1$ from p_1, $n - k$ of them have a distance $k + 1$ from p_1, and only one of them has a distance k from p_1. That is, the relationship of the two end nodes of P_{h+1} can be determined from the relationship of the two end nodes of P_h. More formally, let us define the sequence of vectors $\{a_{i,0}, a_{i,1}, \cdots, a_{i,n}\}$ such that

$$a_{1,0} = 2^n, \; a_{1,i} = 0, \qquad 1 \leq i \leq n,$$

$$a_{k,j} = (n - j + 1)a_{k-1,j-1} + a_{k-1,j} + (j + 1)a_{k-1,j+1}, \qquad k \geq 2, \qquad 1 \leq j \leq n - 1,$$

$$a_{k,0} = a_{k-1,0} + a_{k-1,1},$$

$$a_{k,n} = a_{k-1,n-1} + a_{k-1,n}, \qquad k \geq 2.$$

Note that this sequence of vectors is so defined that $a_{i,j}$ is the number of acceptable labelings of P_i with Q_n in which the distance between the two processors assigned to the two end nodes in P_i is j. Using this sequence, we have the following lemma which determines $N(C_h, Q_n)$.

Lemma 3:

a) $N(P_h, Q_n) = \sum_{i=0}^{n} a_{h,i} = 2^n(n + 1)^{h-1}$.

b) $N(C_h, Q_n) = a_{h,0} + a_{h,1}$.

Proof: The acceptable labelings of P_h with Q_n, in which the distance between the two processors assigned to the two end nodes in P_h is j, come from the following three cases.

Case 1: Adding one more node to P_{h-1}, in which the distance between the two processors assigned to the two end nodes is $j - 1$.

Case 2: Adding one more node to P_{h-1}, in which the distance between the two processors assigned to two end nodes is j.

Case 3: Adding one more node to P_{h-1}, in which the distance between the two processors assigned to two end nodes is $j + 1$.

For these three cases there are $n - j + 1$, 1, and $j + 1$ possible P_{h-1}'s, respectively. Thus, part a) follows from the definition of the sequence of vectors, $\{a_{i,0}, a_{i,1}, \cdots, a_{i,n}\}$.

When n = 2	Δ_0	Δ_1	Δ_2	$N(P_i, Q_2)$
P_1	4			4
P_2	4	8		12
P_3	12	16	8	36
P_4	28	56	24	108
P_5	84	160	80	324

When n = 3	Δ_0	Δ_1	Δ_2	Δ_3	$N(P_i, Q_3)$
P_1	8				8
P_2	8	24			32
P_3	32	48	48		128
P_4	80	240	144	48	512
P_5	320	768	768	192	2048

Fig. 5. Examples of $N(P_i, Q_n)$ for $n = 2$ and $n = 3$.

TABLE I
THE NUMBER OF ACCEPTABLE LABELINGS IN VARIOUS CASES

G_T \ G_P	arbitrary	R_n	Q_n
K_h	$\sum_{i=1}^{r} \lvert H_i \rvert^h - \lvert B_i \rvert^h$ †	$\sum_{i=1}^{r} \lvert H_i \rvert^h - \lvert B_i \rvert^h$	$2^{n+h-1}n - 2^n(n-1)$
S_h	$\sum_{i=1}^{\lvert V_P \rvert} (d_i + 1)^{h-1}$ ††	$\lvert V_P \rvert (n+1)^{h-1}$	$2^n (n+1)^{h-1}$
tree	$W(Y_i)$ †††	$\lvert V_P \rvert (n+1)^{\lvert V_T \rvert - 1}$	$2^n (n+1)^{\lvert V_T \rvert - 1}$
arbitrary	lower bound = $\sum_{i=1}^{r} \lvert H_i \rvert^{\lvert V_T \rvert} - \lvert B_i \rvert^{\lvert V_T \rvert}$		upper bound = $\sum_{i=1}^{\lvert V_P \rvert} (d_i + 1)^{\lvert V_T \rvert - 1}$

† H_i, B_i, $1 \le i \le r$, are, respectively, the component and redundance sets of G_P.
†† $\{d_i\}$ is the degree sequence of G_P.
††† n_i is the arbitrary node in G_T.

Clearly, C_h can be obtained by adding one more edge between the two end nodes of P_h. Thus, part b) follows.Q.E.D.

Fig. 5 shows how to determine $N(P_i, Q_n)$ and thus, $N(C_i, Q_n)$ with the above method when $n = 2$ and $n = 3$, respectively. The entry in row P_i and column Δ_j, denoted by $a_{i,j}$, means the number of acceptable labelings of P_i with Q_n in which the distance between the processors assigned to two end nodes of P_i is j. It can be easily verified that the numbers $N(P_i, Q_n), i \ge 1$, agree with the result of Corollary 2.2. It is interesting to analyze the complexity of calculating $N(C_h, Q_n)$. From Fig. 5 and the recursive definition of $\{a_{i,0}, a_{i,1}, \cdots, a_{i,n}\}$, it requires two multiplications and two additions to determine each entry in row P_i of Fig. 5 from entries in row P_{i-1}, and there are $n + 1$ entries in each row, meaning that the complexity of calculating all entries in a row is $O(n)$. Therefore, the complexity of obtaining $N(C_h, Q_n)$ is $O(nh)$.

Notice that even in the restricted case of $G_P = Q_n$ and $G_T = C_h$, we have to appeal to some nontrivial recursive formula. Naturally, the difficulty in determining $N(G_T, G_P)$ increases with the irregularity of the graphs involved. The main results in this section are summarized in Table I.

IV. APPLICATION, REMARKS, AND EXTENSION

In this section, three application examples are presented to demonstrate the utility of the knowledge of $N(G_T, G_P)$. In light of these examples, some remarks are also made to indicate the complexity of the task assignment problem. More importantly, our results on $N(G_T, G_P)$ are extended to the completely general case (i.e., those assignments with dilations greater than one) in which two related tasks in G_T can be assigned to *any* two processors in G_P (which are not required to be adjacent to each other).

A. Application Examples

Example 1: Consider the processor graph G_P and task graph G_T shown in Fig. 6(a) and (b), respectively. Let $G_P \backslash (i, j)$ denote the graph resulting from the removal of the edge (i, j) from G_P. Then, using the results in Section III, we can calculate $N(G_T, G_P) = 727$, $N(G_T, G_P \backslash (1, 3)) =$

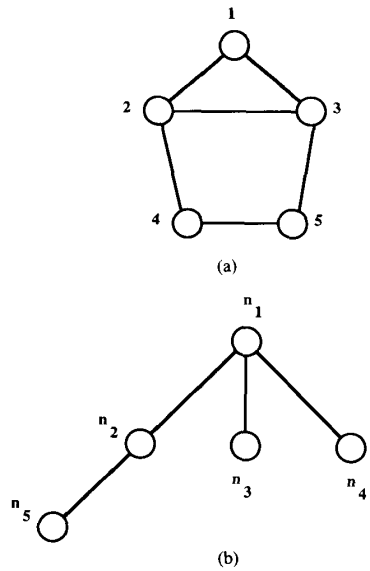

Fig. 6. Example processor and task graphs. (a) Processor graph G_P. (b) Task graph G_T.

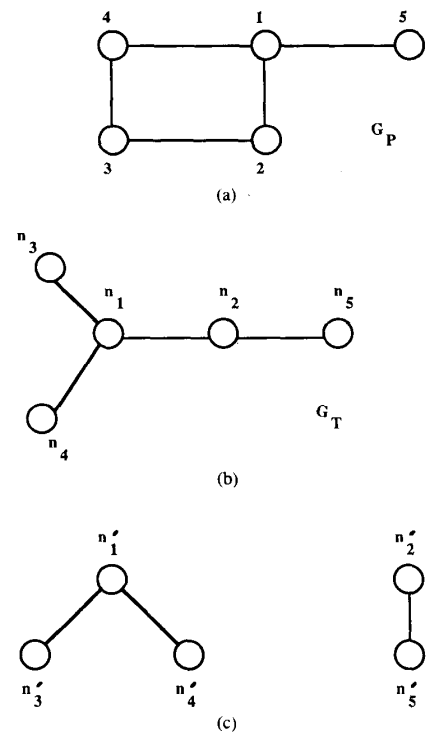

Fig. 7. An example network. (a) Processor graph. (b) Task graph. (c) Decomposed task graph.

477, $N(G_T, G_P\backslash(4, 5)) = 563$, $N(G_T, G_P\backslash(2, 3)) = 405$, and $N(G_T, G_P\backslash(3, 5)) = 489$. Therefore, as far as the number of acceptable assignments is concerned, the edge $(2, 3)$ is the most critical since its removal will cause the largest decrease in the number of acceptable labelings.

Consider the case when a new edge or communication link is to be added in G_P. Let $G_P + (i, j)$ denote the graph resulting from the addition of an edge between nodes i and j in G_P. Then, we obtain $N(G_T, G_P + (1, 4)) = 1091$ and $N(G_T, G_P + (3, 4)) = 1203$. This implies a higher increase in the number of acceptable labelings by adding edge $(3, 4)$ than by adding edge $(1, 4)$. Thus, using $N(G_T, G_P)$ one can determine the edge to be added for a maximal increase in the number of acceptable labelings.

Example 2: Consider the example processor graph G_P and task graph G_T shown in Fig. 7(a) and (b), respectively. Again applying the procedure in Section III, we can construct an *enumeration tree* as shown in Fig. 8, where the number in the square associated with each node represents the number of all acceptable labelings subject to the partial labeling made already for the node and its predecessors.

Notice that the numbers in the squares in the first level (i.e., 192, 90, 81, 90, and 24) are $D_1(i)$, $1 \leq i \leq 5$, and those in the second or lower levels can be obtained by properly decomposing the task tree and multiplying together the numbers of acceptable labelings in each subtree subject to the partial labeling made already for the corresponding node and its predecessors. For example, to determine the number in the square of the node $(n_2, 2)$ whose immediate predecessor is $(n_1, 1)$, we decompose the task tree into two subtrees as shown in Fig. 7(c) and then obtain the number associated with the node $(n_2, 2)$ from $D_{1'}(1) \times D_{2'}(2) = 16 \times 3 = 48$.

The enumeration tree can also be used to determine the number of conditional acceptable labelings, $N(G_T, G_P|P)$, where $P \subseteq \{(t, p)|t \in V_T, p \in V_P\}$. For example, if $P = \{(n_1, 5), (n_3, 1)\}$, then $N(G_T, G_P|P) = 8 + 4 = 12$. This means that there are 12 acceptable labelings in which tasks n_1 and n_3 are assigned to processors 5 and 1, respectively. Moreover, in the state-space search of the task assignment problem, the enumeration tree provides a good indication for the search status of the current node and can be applied to establish a *guided* search. When the computation cost for the heuristic function is high,[3] we can skip some evaluation steps and choose a search route toward an ampler state-space without computing the heuristic function of every offspring. For example, in Fig. 8 the node $(n_1, 1)$ will be chosen since it has the highest potential (i.e., $192 > 90, 81, 24$) for containing the optimal solution. This approach is actually based on the fact that a larger number of acceptable labelings implies that unassigned tasks have a better chance to be spread out in the network.

Clearly, when the goal of achieving load balancing is more important than that of reducing the interprocessor communication cost, the likelihood of making a successful guess with the knowledge of $N(G_T, G_P)$ will be increased. This guided search holds practical importance, since it requires much less search cost and is attractive, especially when we want to reduce the expected search cost and there are many acceptable goal nodes in the state-space.

Example 3: In the state-space search, we naturally want to reduce the number of expanded and generated nodes in the worst case [8]. Consider the enumeration tree in Fig.

[3] For example, the algorithm A^* used in [4] requires a large number of evaluations of a complex heuristic function.

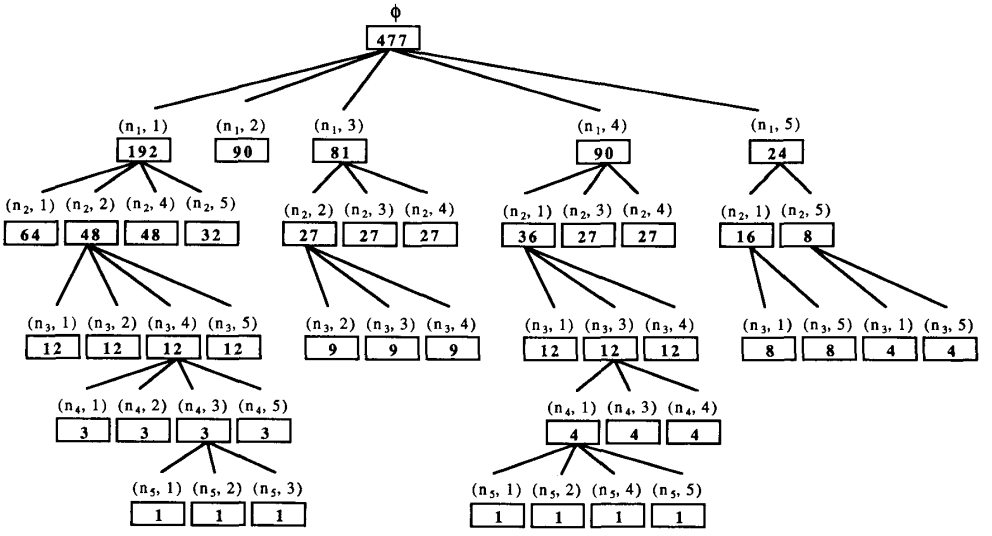

Fig. 8. Part of the enumeration tree with the associated graphs in Fig. 7.

8. It is easy to see that the number of nodes in the ith level is equal to $N(TR_i, G_P)$ where TR_i is the induced subgraph of G_T with the set of nodes $\{n_j | n_j \in V_T$ and $j \leq i\}$. Note that the total number of nodes in the enumeration tree, $1 + \sum_{i=1}^{|V_T|} N(TR_i, G_P)$, and the number of internal nodes, $1 + \sum_{i=1}^{|V_T|-1} N(TR_j, G_P)$, are, respectively, the number of generated nodes and the number of expanded nodes in the worst case of the state-space search. For the example task and processor graphs in Fig. 7, we get $N(TR_1, G_P) = 5$, $N(TR_2, G_P) = 15$, $N(TR_3, G_P) = 47$, $N(TR_4, G_P) = 153$, and $N(TR_5, G_P) = 477$. That is, there are $1 + \sum_{i=1}^{|V_T|-1} N(TR_i, G_P) = 221$ expanded nodes and $1 + \sum_{i=1}^{|V_T|} N(TR_i, G_P) = 698$ generated nodes in the worst case of the state-space search.

Clearly, while $N(TR_{|V_T|}, G_P)$ is just the number of acceptable labelings, the number $\sum_{i=1}^{|V_T|-1} N(TR_i, G_P)$ closely depends on how we encode the task nodes in G_T with n_i, $1 \leq i \leq |V_T|$. In other words, different encodings of the task tree lead to different enumeration trees which usually have different numbers of internal nodes but the same number of leaves. For example, in the case when the task graph in Fig. 7(b) is encoded as the one in Fig. 9, we have $N(TR_1, G_P) = 5$, $N(TR_2, G_P) = 15$, $N(TR_3, G_P) = 47$, $N(TR_4, G_P) = 147$, and $N(TR_5, G_P) = 477$. The maximal numbers of generated and expanded nodes are then reduced to 692 and 215, respectively.

It can be verified by enumeration that among all the possible encodings for the task tree in Fig. 7(b), the encoding in Fig. 9 is the enumeration tree with the minimal number of internal nodes; it minimizes the number of expanded nodes in the worst case of the state-space search when the processor graph is the one in Fig. 7(a). (Such an encoding is termed the *best encoding*.) Improvement in the worst case of the state-space search is not the only advantage of the encoding with a smaller enumeration tree. Since the goal node in the state-space search must be a leaf, searches in the enumeration tree with less internal nodes are naturally expected to have less than average number of expanded and generated nodes.

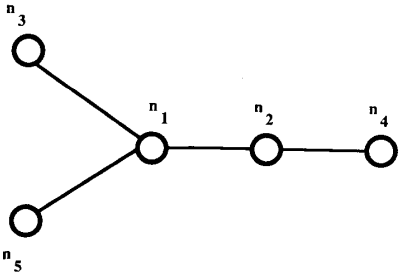

Fig. 9. A different encoding of the task graph in Fig. 7(b).

Using the procedure proposed here, one can construct the enumeration tree for each encoding of the task tree and then determine the best encoding off-line to reduce the computation cost of the state-space search.

B. Remarks

The following remarks are in order to clarify some conjectures which may result from the above examples.

R1. In the first example, the increase of the number of acceptable labelings by adding an edge between two processor nodes with larger degrees may always seem to be greater than that by adding an edge between nodes with smaller degrees. This is not always true. A counterexample is shown in Fig. 10, where G_β and G_γ are obtained by adding edges $(1, 13)$ and $(7, 10)$, respectively, in G_α. Applying the results in Section III, we get $N(P_6, G_\beta) = 10134 < N(P_6, G_\gamma) = 10454$. Note that the degrees of nodes 1 and 13 in G_α are 4, whereas those of nodes 7 and 10 in G_α are 3.

R2. We get $N(P_3, G_\gamma) = 196 > N(P_3, G_\gamma) = 192$ in Fig. 10. This means that the edge $(1, 13)$ in G_δ is more important than the edge $(7, 10)$ when the task graph is P_3, but less important than the edge $(7, 10)$ when the task graph is P_6. This

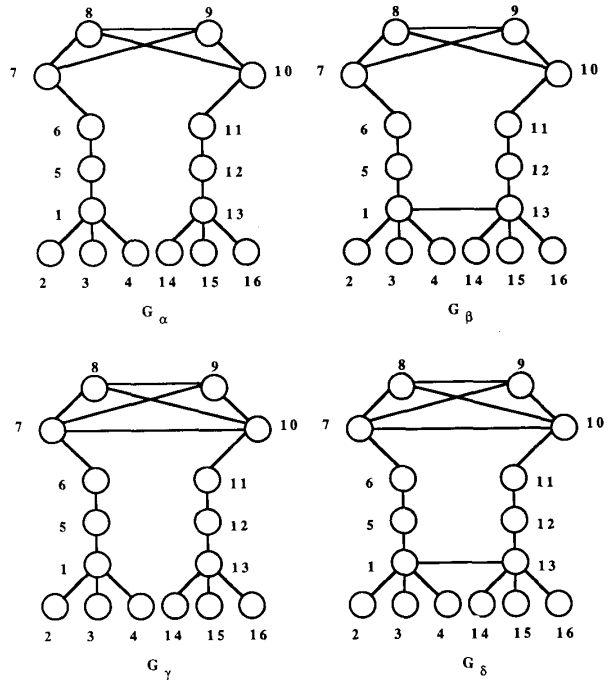

Fig. 10. A counterexample showing that adding an edge between two nodes with larger degrees does not always result in a higher increase in the number of acceptable labelings.

fact not only indicates that the importance of each edge in the processor graph depends on the structure of the associated task graph but also shows that $N(G_{T_1}, G_{P_1}) > N(G_{T_1}, G_{P_2})$ does not imply $N(G_{T_2}, G_{P_1}) > N(G_{T_2}, G_{P_2})$ for $G_{T_2} \neq G_{T_1}$.

R3. One may also conjecture in Example 2 that the processor node which leads to the amplest state-space is always the node with the maximal degree. Again, a counterexample is given in Fig. 11 in which the degree of node 1 in the processor graph is greater than that of node 5 (i.e., $d(1) = 4 > d(5) = 3$), while $D_1(1) = 225 < D_1(5) = 289$.

R4. Consider the two encodings of a task tree in Fig. 12(a) and (b). G_T and G_{T_2} in Fig. 13(a) and (b) are, respectively, the TR_6's corresponding to the encodings in Fig. 12(a) and (b). We then have $N(G_{T_1}, S_4) = 640 > N(G_{T_2}, S_4) = 616$, and $N(G_{T_1}, P_4) = 482 < N(G_{T_2}, P_4) = 484$ where S_4 and P_4 are, respectively, the star and path with four nodes. It can be verified that the encoding of the task tree in Fig. 12(a) is the best encoding when $G_P = P_4$, and on the other hand, the encoding in Fig. 12(b) is the best encoding when $G_P = S_4$. This fact indicates that the best encoding of a task tree depends on the structure of the associated processor graph, i.e., $N(G_{T_1}, G_{P_1}) > N(G_{T_2}, G_{P_1})$ does not always imply $N(G_{T_1}, G_{P_2}) > N(G_{T_2}, G_{P_2})$ for $G_{P_2} \neq G_{P_1}$.

As can be seen from the above remarks, the task assignment problem is more complicated than it may appear to be. This is the very reason that a rigorous procedure like the one treated in this paper must be called for.

C. Extension

Thus far, we dealt with only those task assignments with dilations not greater than one. However, our results developed

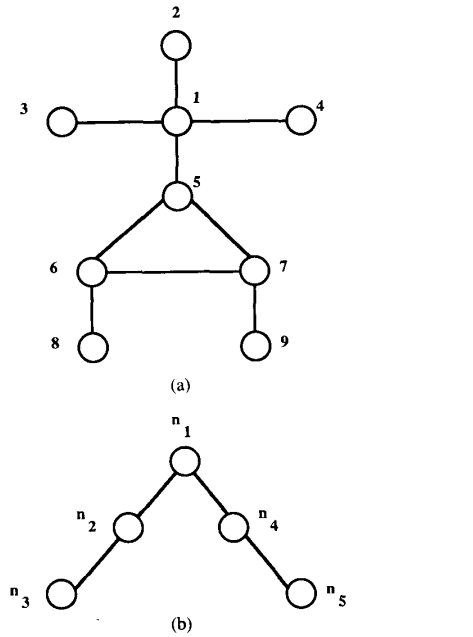

Fig. 11. An example network for Remark 3 where $D_1(1) < D_1(5)$ and $d(1) = 4 > d(5) = 3$. (a) Processor graph. (b) Task graph.

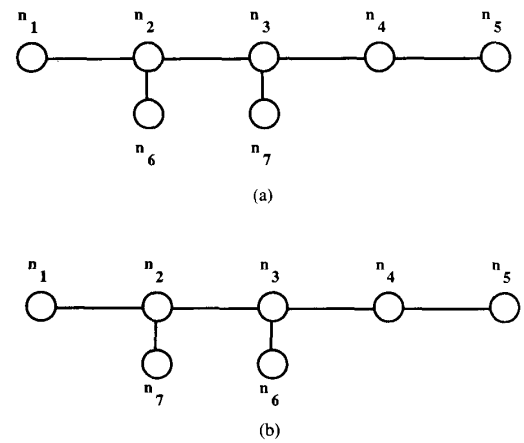

Fig. 12. Example of different encodings which are associated with different enumeration trees. (a) An encoding of a task tree. (b) Another encoding of a task tree.

for the case of the dilation not greater than one can be extended to the completely general case, in which the dilation can be greater than one. Suppose the allowable dilation (AD) is a positive integer $k > 1$. Then, a *communication graph* G_C can be obtained from the processor graph G_P in such a way that $V_C = V_P$ and every pair of processor nodes in G_C is connected iff the number of hops between the pair of processor nodes in G_P is less than or equal to k. For example, given the processor graph in Fig. 1(a) and AD = 2, we have the communication graph in Fig.14(b) where a solid line means a one-hop communication and a dashed line denotes a two-hop communication. Clearly, when AD = 1, the communication graph is the same as the processor graph.

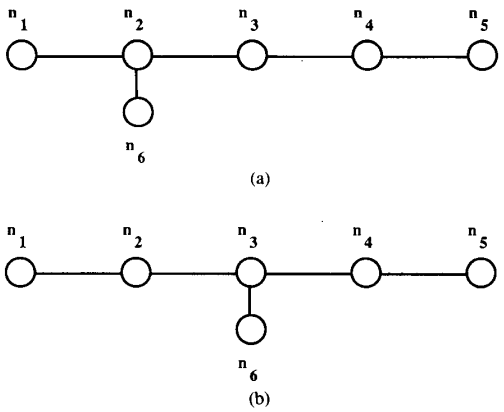

Fig. 13. TR_6 according to the encoding in Fig. 12. C_{T_1}, TR_6 according to Fig. 12(a). C_{T_2}, TR_6 according to Fig. 12(b).

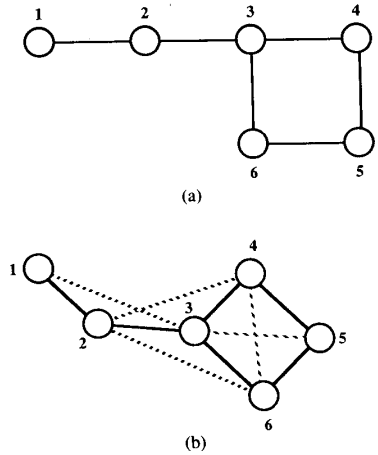

Fig. 14. Determination of a communication graph. (a) Processor graph G_P. Communication graph G_C.

Notice that the constraint "every two related tasks in G_T must be assigned to either a single processor or two processors, the distance between which is less than or equal to k in G_P" is equivalent to the constraint "every two related tasks in G_T must be assigned to either a single processor or two adjacent processors in G_C." Then, we can treat the task assignment problem with processor graph G_P and AD > 1 as the task assignment problem with processor graph G_C and AD $= 1$. By substituting the communication graph G_C for the processor graph G_P in our previous results, we can formulate $N(G_T, G_C)$ instead of $N(G_T, G_P)$. Following exactly the same procedures in previous examples, we can apply this knowledge to provide a simplified state-space search (as in Example 2) and reduce the size of the state-space to be searched (as in Example 3).

V. Conclusion

In this paper, we have derived the bounds for the number of acceptable task assignments for arbitrary G_T and G_P, a recursive formula for the case when G_T is a tree, and closed-form expressions for more restricted cases. Notice that the knowledge of $N(G_T, G_P)$ can be applied not only for improving the state-space search of the task assignment problem but also for evaluating the importance of each system component when it is desired to have more choices in assigning tasks. By comparing the number of acceptable assignments before and after removing a certain node/link in G_P, the importance of the node/link can be evaluated. Furthermore, we have extended the results on $N(G_T, G_P)$ to the completely general case (i.e., those assignments with dilations greater than one) in which two related tasks in G_T can be assigned to any two processors in G_P.

Unfortunately, the general formula for an arbitrary G_T could not be derived. As shown in Section III-C, even in the restricted case when G_T is a cycle and G_P is a hypercube, we have to appeal to a nontrivial recursive formula. Clearly, the difficulty associated with the problem increases with the irregularity of the graphs involved. Unlike the isomorphic mapping on which conventional approaches are based [14], the mapping under the adjacency requirement allows for many-to-one mappings. Such a graph mapping is of practical importance, since the compromise between exploiting the parallelism and minimizing the communication cost is an important design problem.

Appendix

List of Symbols

G_P	A processor graph.
G_T	A task graph.
G_C	A communication graph.
V_α	The set of nodes in the graph G_α.
E_α	The set of edges in the graph G_α.
$H_i(B_i), 1 \le i \le r$	Components (redundance sets) of a processor graph.
$N(G_T, G_P)$	The total number of assignments which satisfy the adjacency condition.
U_m	An m-dimensional vector all entries of which are one.
$f_A(V)$	The attaching function of the vector V associated with the adjacency matrix A.
$\odot$	The product of vectors.
$*_A$	The multiplication of vectors associated with adjacency matrix A.
V^*	The sorted vector of the vector V.
$W(V)$	The weight function of the vector V.
$C(n_i)$	The set of children of a node n_i in a rooted tree.
$T(n_i)$	The tree formed by the node n_i and its descendants.
Y_i	The carrying vector of a node $n_i \in V_T$.
D_i	The distribution vector of a node $n_i \in V_T$.

References

[1] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Comput. Mag.*, vol. 13, pp. 57-69, Nov. 1980.

[2] W. W. Chu and L. M.-T. Lan, "Task allocation and precedence relations for distributed real-time systems," *IEEE Trans. Comput.*, vol. C-36, no. 6, pp. 667-679, June 1987.

[3] C. L. Seitz, "The Cosmic Cube," *Commun. ACM*, vol. 28, no. 1, pp. 22–33, Jan. 1985.
[4] C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Trans. Comput.*, vol. C-34, no. 3, pp. 197–203, Mar. 1985.
[5] F. Harary, "The topological cubical dimension of a graph," in *Lecture Notes First Japan Conf.. Graph Theory Appl.*, June 3, 1986.
[6] R. C. Read and D. G. Corneil, "The graph isomorphism disease," *J. Graph Theory*, pp. 339–363, 1977.
[7] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[8] N. J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.
[9] S. H. Bokhari, "On the mapping problem," *IEEE Trans. Comput.*, vol. C-30, no. 3, pp. 207–214, Mar. 1981.
[10] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Comput. Mag.*, vol. 15, pp. 50–56, June 1982.
[11] G. S. Rao, H. S. Stone, and T. C. Hu, "Assignment of tasks in a distributed processing system with limited memory," *IEEE Trans. Comput.*, vol. C-28, no. 4, pp. 291–299, Apr. 1979.
[12] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1969.
[13] A. W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Its Applications*. New York: Academic, 1969.
[14] D. G. Corneil and D. G. Kirkpatrick, "A theoretical analysis of various heuristics for the graph isomorphism problem," *SIAM J. Comput.*, vol. 9, no. 2, pp. 281–297, May 1980.

Kang G. Shin (S'75–M'78–SM'83), for a photograph and biography, see this issue, p. 18.

Ming-Syan Chen (S'88–M'88), for a photograph and biography, see this issue, p. 18.