Task Allocation in Distributed Systems: A Split Graph Model

Hesham H. Ali, Hesham El-Rewini

Department of Mathematics and Computer Science
University of Nebraska at Omaha
Omaha, NE 68182-0243

Abstract. The problem of task allocation in distributed systems has been studied by many researchers. Several approaches have been used to model and study the problem including integer programming, heuristic methods, and graph theoretic models. These approaches considered only restricted forms of the general problem. In this paper, we introduce a new model to represent the problem of allocating tasks on heterogeneous distributed systems. The model consists of a complete split graph that represents the communication cost among tasks as well as the execution cost of each task on the system processors. This model allows the incorporation of various constraints into the allocation problem. We show that the task allocation problem is equivalent to the problem of weighted clique partitioning in complete split graphs, which we proved to be NP-complete. We present a clique partitioning algorithm that employs the properties of split graphs for solving the problem in its general form. We show that the algorithm generates optimal solutions in some cases, while performing fairly well in general.

1. Introduction.

The fast progress of large integration technology has made distributed computing systems economically attractive for many computer applications. Distributed computing provides facility for remote computing resources and data access. Besides their flexibility, reliability, and modularity, distributed systems can be used for parallel processing. However, several problems have slowed down the widespread use of distributed systems. A major problem is the degradation in the system throughput caused by the saturation effect. In an ideal multiple processor environment, we expect the throughput to increase linearly as the number of processors increase. In practice, the throughput increases significantly only for the first few additional processors but when the number of processors starts to increase, the throughput does not increase accordingly. This saturation effect is usually caused by excessive interprocessor communication by messages transferred from one task to another residing on different processors. In order to minimize or avoid the saturation effect, the cost of the interprocessor communication should be considered in allocating system tasks to the processing elements [6].

In the assignment of tasks to processors there are two types of cost; the cost of execution of a task on a processor and the cost of interprocessor communication. In order to improve the performance of a distributed system, two goals need to be met: 1) interprocessor communication has to be minimized and 2) the execution cost needs to be balanced among different processors. These two goals

JCMCC 14 (1993), pp. 15-32

seem to conflict with one another. On one hand, having all tasks on one processor will remove interprocessor communication cost but results in poor balance of the execution load. On the other hand, an even distribution of tasks among processors will maximize the processor utilization but might also increase interprocessor communication. Thus, the purpose of a task allocation technique is to find some task assignment in which the total cost due to interprocessor communication and task execution is minimized.

Several approaches to the problem of task assignment in distributed system have been developed. They can be roughly classified into three categories, namely, graph theoretic [3, 4, 12-14], mathematical programming [6, 7, 13] and heuristic methods [5, 10-12]. The graph theoretic approach applies the max-flow min-cut algorithm on a graph that represents the problem to get task assignment with minimum interprocessor communication. In the mathematical programming approach, the task assignment problem is formulated as an optimization problem and mathematical programming techniques are used to provide a task assignment. Since the graph theoretic approach employs the max-flow min-cut algorithm, the graph models are effective when the distributed system consists of two processors only. These models fail when the system has more than two processors. Also, this approach uses a restricted objective function that considers only the interprocessor communication cost. Mathematical programming techniques imposes several assumptions that limits their effectiveness and can not be applied to real world systems.

In this paper, we introduce a graph theoretic approach that models the task allocation problem in distributed systems with any number of processors. The new approach allows the incorporation of various constraints into the task allocation model. We also use several properties of split graphs to devise a weighted clique partitioning heuristic for solving the task allocation problem. To justify solving the problem heuristically, we show that the task allocation problem is equivalent to the problem of partitioning a complete split graph into a set of maximum weighted disjoint cliques, which we show to be NP-complete. This paper is organized as follows. In Section 2, we give the necessary definitions of split graphs and distributed systems. The split graph model is introduced in Section 3. In Section 4, we discuss the task allocation problem when the optimization criterion is to minimize the total execution cost plus the total communication cost. We also show that the task allocation problem is equivalent to the problem of weighted clique partitioning in complete split graphs which is NP-complete. Section 5 includes the clique partitioning heuristic and its performance analysis. The experimental results are given in Section 6. We summarize our conclusions in Section 7.

2. Preliminaries.

2.1 Split graphs

An undirected graph G is defined by two sets; a set of nodes V and a set of edges E. A set of nodes $U \subseteq V$ is called an independent (stable) set if its elements are

pairwise non-adjacent, that is, there exists no edges between its nodes. Similarly, a set of nodes $U \subseteq V$ is said to be a complete set (clique) if its elements are pairwise adjacent, that is, there is an edge between every pair of its nodes.

An undirected graph $G = (V, E)$ is defined to be split if there is a partition $V = V_1 \cup V_2$ of its vertex set into an independent set V_1 and a complete set V_2. There is no restriction on edges between vertices of V_1 and vertices of V_2. A split graph is said to be a complete split graph if there is an edge between every node in V_1 to every node in V_2. We use the term *cross edge* to denote an edge that connects a node in V_1 to a node in V_2. The edges connecting nodes within the complete set are denoted by *complete edges.* An example of a complete split graph is shown in Figure 1 in which $|V_1| = 3$ and $|V_2| = 4$.

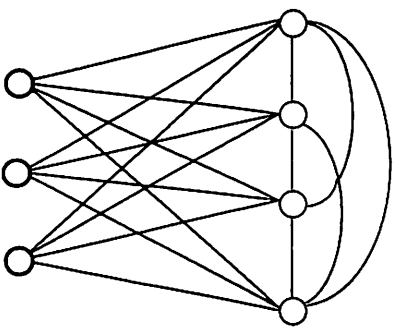

Figure 1. Complete Split Graph

There are two main graph theoretic properties of split graphs. Since a stable set of G is a complete set of the complement graph G and vice versa, the complement of any split graph is also a split graph. Also, due to the fact that V_2 is a complete set, a split graph can not contain any chordless cycles of size four or more. In other words, split graphs form a special class of chordal (triangulated) graphs [9]. These two properties form another characterization of split graphs. A graph G is a split graph if and only if it is chordal and its complement is also chordal.

2.2 Distributed systems

A distributed system is assumed to be made up of an arbitrary number m of heterogeneous processing elements. These processing elements are assumed to be interconnected via a communication network. System load consists of n separate cooperating and communicating modules called tasks. Each task requires a certain amount of computation expressed by its execution time. Since the processing elements are heterogeneous, the execution time of the same task might differ from one processing element to another. A pair of tasks might also need to exchange a certain amount of information and, hence, allocating them on different processing elements might result in a cost expressed by their communication cost.

3. The split graph model.

The problem of allocating tasks to processors in distributed systems can be modeled by split graphs. A set of processors in a distributed system can be modeled by the independent set V_1 in the split graph $G = (V_1 \cup V_2, E)$. The set of tasks can be represented by the complete set V_2 in the graph. A complete edge (T_i, T_j) is used to represent the intercommunication between tasks T_i and T_j. The weight on the edge expresses the communication cost between the two tasks when they are assigned to two different processors. The weight on a cross edge (T_k, P_ℓ) represents the execution cost of task T_k on processor P_ℓ. A complete edge (T_i, T_j) with a weight of zero implies that there is no communication between the tasks T_i and T_j. It follows that a complete split graph is constructed in this process to represent the distributed system as shown in Figure 2.

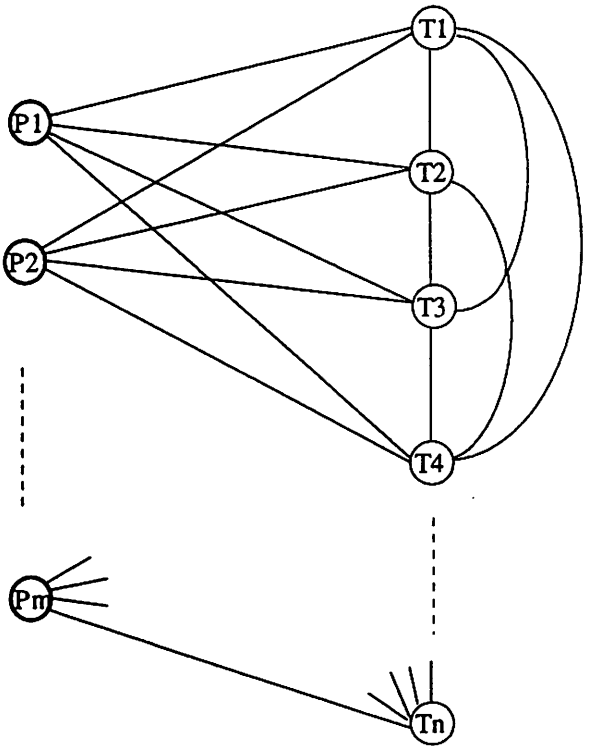

Figure 2. Split Graph Model

Using this model, a task assignment corresponds to some graph partitioning of the split graph. Each graph partition consists of a node $p \in V_1$ from the independent set and a set of nodes $U \subseteq V_2$ from the complete set. The set U represents the set of tasks assigned to the processor represented by the node p. It follows that

each partition forms a complete set (clique) which implies that task allocation is equivalent to partitioning the split graph into node-disjoint cliques, as shown in Figure 3.

An optimization criterion is used to measure how good a task assignment is. The objective function is a function of the weights on both cross and complete edges of the resulted cliques. Possible optimization criteria include minimizing completion time, minimizing the interprocessor communication cost, balancing the overall load of the system or minimizing both the communication cost and execution cost. Each objective function corresponds to finding a clique partitioning of the split graph satisfying certain properties. For example, when the objective is to minimize the communication cost, partitioning the graph into cliques with maximum total weight of the complete edges will result in the optimal solution.

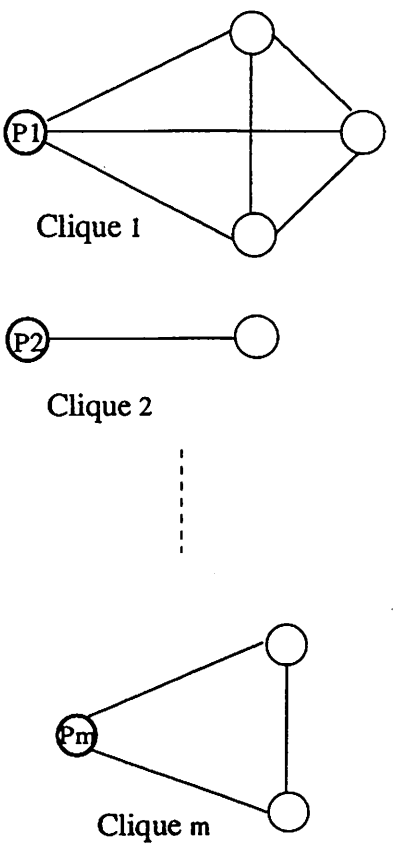

Figure 3. Clique Partitioning in the Split Graph Model

4. Minimizing the total cost.

One of the most common optimization criterion in task allocation in distributed systems is to minimize the total cost. The total cost can be expressed as the summation of the total execution cost and the total interprocessor communication cost. Using the split graph model introduced in the previous section, the total cost of each task assignment can be obtained as follows. The total execution cost is equal to the summation of the weights on all the cross edges that belong to any of the cliques. Recall that each assignment corresponds to partitioning the split graph into a set of cliques. Also, the communication cost can be measured by the summation of the weights on the complete edges that do not belong to any of the cliques.

Suppose a task assignment corresponds to partitioning the split graph $G = (V, E)$ into a set of node-disjoint cliques $\{C_1, C_2, \ldots, C_m\}$ where m is the size of the independent set V_1, and each clique C_i contains a node p_i that represents a processor in the system. For each task assignment $V = C_1 \cup C_2 \cup \ldots \cup C_m$, let us define the following:

- A cross edge (v_a, p_i) is called an active cross edge if $v_a \in C_i$ and inactive cross edge otherwise.
- A complete edge (v_a, v_b) is called an active complete edge if both $v_a, v_b \in$ the same clique C_i and inactive complete edge otherwise.

Now, the total cost of the task assignment can be obtained as follows.

$EC =$ sum of the weights of all active cross edges.

$CC =$ sum of the weights of all inactive complete edges.

$TC = EC + CC.$

In order to minimize the total cost, the clique partitioning should be obtained in such a way that maximizes the weights on the active complete edges while minimizes the weights on the active cross edges. If we replace the weight on each cross edge $w(v_a, p_i)$ by $(\beta - w(v_a, p_i))$, where β is a large positive integer, then the objective will be to maximize the weights on the active cross edges as well as the weights on the active complete edges. It follows that the problem of minimizing the total cost of task allocation in a distributed system is equivalent to the problem of partitioning the corresponding split graph into cliques that contain the edges with the maximum total weight, or the maximum weighted cliques partitioning problem. The two problems can be defined as follows.

Task allocation in distributed systems:

Given a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$, a set of processors $P = \{p_1, p_2, \ldots, p_m\}$, communication matrix C, where C_{ij} = cost of communication between t_i and t_j if assigned to different processors, execution matrix X, where X_{ij} = cost of executing task t_i on processor p_j, and a positive integer A, is there an assignment of

tasks T to processors P such that $EC + CC \leq A$, where $EC = \Sigma X_{ij}$, task t_i is assigned to processor p_j, for $1 \leq i \leq n$, $1 \leq j \leq m$ and $CC = \Sigma C_{ij}$, task t_i and task t_j are assigned to different processors for $1 \leq i \leq n$, $1 \leq j \leq n$?

Weighted clique partitioning in complete split graphs:

Given a complete split graph $G = (V, E)$, $V = V_1 \cup V_2$, where V_1 is an independent set and V_2 is a complete set, weights $\ell(e) \in I$, where I is the set of integers, for each $e \in E$, and a positive integer B, is there a partition of V into exactly m disjoint cliques $C_1, C_2, \ldots, C_m$, $m = |V_1|$ such that $\sum_{e \in C_i} \ell(e) \geq B$ for $1 \leq i \leq m$?

The proof of the following lemma follows directly from the previous discussion.

Lemma 1. *The problem of task allocation in distributed systems is equivalent to the problem of weighted clique partitioning in complete split graphs.*

In the following theorem, we prove that the weighted clique partitioning in complete split graphs is NP-complete which will imply that the task allocation problem is also NP-complete [1].

Theorem 1. *The weighted clique partitioning in complete split graphs problem is NP-complete.*

Proof: We prove the theorem using a transformation from the graph partitioning problem. The graph partitioning problem can be described as follows:

Graph partitioning

Given a graph $H = (V', E')$, weights $w'(v) = 1$ for each $v \in V'$, $\ell'(e) \in Z^+$ for each $e \in E'$, and positive integers K and J, is there a partition of V' into disjoint sets $V'_1, V'_2, \ldots, V'_{m'}$ such that $\sum_{v \in V'_i} w'(v) \leq K$ for $1 \leq i \leq m'$ and if $E'' \subseteq E'$ is the set of edges that have their two end points in two different sets V'_i, then $\sum_{e \in E''} \ell'(e) \leq J$?

This problem is known to be NP-complete for any fixed $K \geq 3$ even when $\ell'(e) = 1 \forall e \in E'$ [8]. Given an instance of the graph partitioning problem (GPP), we can construct an instance of the weighted clique problem (WCP) as follows:

- V_1 is a set of independent nodes such that $|V_1| = \binom{n}{K}$,
- $V_2 = V'$,
- $E = E_1 \cup E_2 \cup E_3$, where $E_1 = E'$, $E_2 = \{(u, v) : u, v \in V_2, (u, v) \notin E_1\}$, $E_3 = \{(u, v) : u \in V_1, v \in V_2\}$,
- $\ell(e) = \ell'(e) \forall e \in E_1$,
- $\ell(e) = 0 \forall e \in E_2$,
- $\ell(e) \in \{1, \alpha\} \forall e \in E_3$, where α is a small negative integer $\leq -(\sum_{e \in E_1} \ell(e) + n)$. The weights $\ell(e)$, $e = (u, v_i)$, $1 \leq i \leq n$, where $e \in E_3$ are represented as a weight vector $\langle w_1, w_2, \ldots, w_n \rangle$, where $w_i \in \{1, \alpha\}$ and $\ell((u, v_i)) = w_i$. The tabulation given below shows the $\binom{n}{K}$ weight vectors for all the edges in E_3. Notice that in each weight vector, there are

exactly K 1's and $n-K$ α's. This weight assignment guarantees that for any set of nodes $U \subseteq V_2$, $|U| \leq K$, there exists a node $v \in V_1$ such that $\ell((u,v)) = 1$ for all $u \in U$.

$$\leftarrow K \rightarrow \leftarrow n-K \rightarrow$$

$$\begin{array}{c} 11\ldots1\alpha\alpha\ldots\ldots\alpha \\ 1\ldots1\alpha1\alpha\ldots\ldots\alpha \\ 1\ldots1\alpha\alpha1\ldots\ldots\alpha \\ \ldots\ldots \\ \alpha\alpha\ldots\ldots\alpha11\ldots1 \end{array}$$

$$\leftarrow n-K \rightarrow \leftarrow K \rightarrow$$

- $B = \sum_{e \in E_1} \ell(e) + n - J$, or $B = \sum_{e \in E'} \ell'(e) + n - J$.

Now the claim is that a positive (negative) answer to the question of WCP implies a positive (negative) answer to the question of GPP. A positive answer to the question of WCP implies that there exists a partition of V into disjoint cliques $C_1, C_2, \ldots, C_m$ such that $\sum_{e \in \text{ any } C_i} \ell(e) \geq B$. Since B is a positive integer and α is a very small negative integer, none of the cliques should contain an active cross edge e, such that $\ell(e) = \alpha$. Since there are exactly n edges with weights equal one that are in the cliques and belong to E_3, it follows that $\sum_{e \in \text{ any } C_i \& e \notin E_3} \ell(e) \geq B - n$. Given that $C_i = (V(C_i), E(C_i))$, we can use $V_i' = V(C_i) - \{v\}$, where $v \in V_1$, that is, each subgraph in the solution of GPP is one of the cliques provided by the solution of WCP minus the node that belongs to V_1 along with all the edges that belong to E_2 or E_3. Formally, each subgraph is defined as $G_i' = (V_i', E_i')$, where $V_i' = V(C_i) - \{v\}$, $v \in V_1$, and $E_i' = E(C_i) - (E_2 \cup E_3)$. It follows that $\sum_{e \in \text{ any } G_i' \& e \in E'} \ell'(e) \geq B - n$. Since

$$\sum_{e \notin \text{ any } G_i'} \ell'(e) = \sum_{e \in E'} \ell'(e) - \sum_{e \in \text{ any } G_i'} \ell'(e),$$

it follows that

$$\sum_{e \notin \text{ any } G_i'} \ell'(e) \leq \sum_{e \in E'} \ell'(e) - (B - n).$$

$$\Leftrightarrow \sum_{e \notin \text{ any } G_i'} \ell'(e) \leq \sum_{e \in E'} \ell'(e) - \left(\sum_{e \in E'} \ell'(e) + n - J - n \right)$$

$$\Leftrightarrow \sum_{e \notin \text{ any } G_i'} \ell'(e) \leq J$$

$$\Leftrightarrow \sum_{e \in E''} \ell'(e) \leq J.$$

Now it is left to prove that for each subgraph G'_i, $\sum_{v \in V'_i} w'(v) \leq K$. Since $\alpha \leq -n - \sum_{e \in E_1} \ell(e)$, it is guaranteed that none of the cliques contains an edge e, such that $\ell(e) = \alpha$. It follows that only the cross edges e, such that $\ell(e) = 1$ will be included in the cliques. Since every clique includes a node in V_1 by definition and that for each node in V_1 , there are exactly K incident edges with unit weight, it follows that the number of nodes in each clique is always less than or equal to $K + 1$. In other words, at most K nodes from V_2 would belong to any clique. It follows that at most K nodes would belong to any partition G'_i. Since $w'(v) = 1$ for all nodes in V', then for each partition G'_i, the number of nodes is less than or equal to K, or $\sum_{v \in V'_i} w'(v) \leq K$.

Clearly, it can be shown that a negative answer to the decision problem WCP implies a negative answer to the decision problem GPP. Since the transformation is polynomial, it follows that the WCP is NP-hard. Also, since a given solution to the problem can be verified in polynomial time, it follows that the WCP is in NP and hence is NP-complete. Since problem GPP remains NP-complete even when $\ell'(e) = 1 \; \forall e \in E'$ [8], the WCP also remains NP-complete even when the weights $\ell((u,v)) \in \{0,1\} \; \forall u, v \in V_2$. ∎

The complexity of the task allocation problem in distributed systems can be determined from Lemma 1 and Theorem 1. It follows that the decision problem is NP-complete and consequently the optimization problem is NP-hard even in the case when the communication cost between tasks allocated on different processing elements is either zero or one.

5. Clique partitioning heuristic.

Having shown that finding an optimal allocation, with the minimum total cost, of tasks to processors of a given distributed system is NP-hard, we introduce a heuristic that handles the general case of the problem. Given an instance of the task allocation problem, we can construct an instance of the weighted clique partitioning problem as follows.

$$V_1 = P, \quad V_2 = T$$
$$\ell(e) = C_{ij}, \text{ for any edge } e = (v_i, v_j) \text{ and } v_i, v_j \in V_2$$
$$\ell(e) = \beta - X_{ij}, \text{ where } e = (v_i, v_j) \text{ and } v_i \in V_2, v_j \in V_1, \beta \text{ is a large positive integer}$$

Clearly, a solution for the clique partitioning problem will provide a solution for the task allocation problem as shown in the previous section. Since it is required to partition the set of nodes into exactly m disjoint cliques, each clique should contain exactly one node that belongs to the independent set V_1. It follows that each clique C_i contains one node $p_i \in V_1$ and x nodes that belong to $V_2, 0 \leq x \leq n$. The cost of assigning each pair of nodes to the same clique is evaluated and compared to the cost of assigning them to different cliques. This comparison affects the decision

whether each pair of nodes should be included in the same clique or not. For every pair of nodes $u, v \in V_2$ and clique C_i, we define the following cost function.

$$F(u,v,C_i) = \{\ell(u,p_j) - \ell(u,p_i)\} + \{\ell(v,p_k) - \ell(v,p_i)\}, \text{ where}$$
$$\ell(u,p_j) \geq \ell(u,p_\ell) \text{ and } \ell(v,p_k) \geq \ell(v,p_\ell),$$
$$1 \leq \ell \leq m \& p_i, p_j, p_k, p_\ell \in V_1,\ p_i \in C_i$$

The function $F(u,v,C_i)$ gives an indication about the appropriateness of assigning nodes u and v to the clique C_i. The larger the value of the function $F(u,v,C_i)$, the less appropriate it will be to assign both u and v to the clique C_i. Having $F = 0$ implies that C_i is the best clique to include both nodes u and v.

We divide the heuristic into two phases: preprocessing and clique partitioning. In the preprocessing phase: 1) m disjoint cliques are initialized such that each clique C_i contains a node $p_i \in V_1$, 2) the complete edges are sorted in descending order according to their weights, and 3) for each edge (u,v) in the sorted list, the function $F(u,v,C_i)$ is evaluated for all values of $1 \leq i \leq m$. In the clique partitioning phase, the split graph is partitioned into a set of m cliques $C_1, C_2, \ldots, C_m$. The sorted complete edges are processed one at a time until all edges have been processed. For each complete edge (u,v), if the weight of the edge is greater than or equal to the cost function $F(u,v,C_i)$ of the two nodes u and v for any clique C_i, the algorithm then recommends the assignment of both nodes to the same clique, the one with the smallest cost function. Similarly, if the weight of the edge (u,v) is less than the cost function F of the two nodes for all cliques, the heuristic will suggests that node u (v) be assigned to the clique containing node p_j (p_k), where $\ell(u,p_j) \geq \ell(u,p_\ell)$ and $\ell(v,p_k) \geq \ell(v,p_\ell)$, $1 \leq \ell \leq m \& p_j, p_k, p_\ell \in V_1$.

In general, the nodes might have been already assigned to one of the cliques, and hence the algorithm compares the new suggestion with the old assignment. There are four possibilities in this case, keeping the original assignment of both nodes, move one of the nodes to another clique, or move both of them to a new clique(s). The relative weight of these four cases are then evaluated and the maximum weight will decide which option is the best among the four cases. The weight of these cases are denoted by $wgt1$, $wgt2$, $wgt3$, and $wgt4$ in the algorithm. The algorithm keeps track of current assignment of each node u and store it in $clique(u)$, which is initialized by zero for all nodes. It follows that at any given instance, $clique(u) = i$ for all nodes u in the clique C_i. If u is one of the endpoints of the edge under processing, $clique(u)$ might change if the algorithm decides to move u to another clique. Notice that this decision is not final, since any of the nodes might be moved to another clique because of another complete edge down in the sorted list. The details of algorithm Partition are given below.

Algorithm Partition:

Input : A complete split graph $G = (V_1 \cup V_2, E_X \cup E_C)$, where V_1 is the independent set, V_2 is the complete set, E_X is the set of cross edges, and E_C is the set of complete edges.

Output : Set of node disjoint cliques $C_1, C_2, \ldots, C_m$.

Preprocessing and Initialization

{ Initialize all cliques }
$C_i \leftarrow \{p_i\} 1 \leq i \leq m, \quad p_i \in V_1$
$C_0 \leftarrow \emptyset$

{ Sort complete edges }
Sort the complete edges E_c in descending order according to the weight

{ Evaluate the function F for every pair of nodes in the complete set }
Evaluate $F(u, v, C_i) \forall (u, v) \in V_2, \quad 1 \leq i \leq m$

{ Initialization }
Let clique(u) $\leftarrow 0 \forall u \in V_2$
Let node(i) be a function that returns $p_i \in V_1$ that is assigned to C_i
Let node(0) $\leftarrow 0$
$l(u, 0) \leftarrow -\infty \forall u \in V_2$

Clique Partitioning

repeat
{ Process a complete package }
Let (u, v) be the unmarked edge with the maximum weight in E_c

{ Comparing the weight on the complete edge and the function F }
if $l(u, v) < F(u, v, C_i), 1 \leq i \leq m$ **then begin**
$p_x \leftarrow p_k$, where $l(u, p_k \geq l(u, p_i), 1 \leq i \leq m$
$p_y \leftarrow p_k$, where $l(v, p_k \geq l(v, p_i), 1 \leq i \leq m$
end
else begin
$p_x \leftarrow p_k$, where $F(u, v, C_k \leq F(u, v, C_i), 1 \leq i \leq m$
$p_y \leftarrow p_x$
end

{ What if u and v remain in their current cliques? }
$wgt1 \leftarrow l(u, \text{node}(\text{clique}(u))) + l(v, \text{node}(\text{clique}(v))) +$
$\sum l(u, t) \forall t \in \text{clique}(u) + \sum l(v, t) \forall t \in \text{clique}(v)$

{ What if u moves to another clique and v does not? }
$wgt2 \leftarrow l(u, p_x) + l(v, \text{node}(\text{clique}(v))) +$
$\sum l(u,t) \forall t \in C_x + \sum l(v,t) \forall t \in \text{clique}(v)$

{ What if v moves to another clique and u does not? }
$wgt3 \leftarrow l(u, \text{clique}(u)) + l(v, p_y) +$
$\sum l(u,t) \forall t \in \text{clique}(u) + \sum l(v,t) \forall t \in C_y$

{ What if both u and v move to other cliques? }
$wgt4 \leftarrow l(u, p_x) + l(v, p_y) +$
$\sum l(u,t) \forall t \in C_x + \sum l(v,t) \forall t \in C_y$

if maximum($wgt1, wgt2, wgt3, wgt4$) = $wgt2$ or $wgt4$ **then begin**
{ u moves to C_x }
Remove u from its current clique
$C_x \leftarrow C_x \cup \{u\}$
clique(u) $\leftarrow x$
end

if maximum($wgt1, wgt2, wgt3, wgt4$) = $wgt3$ or $wgt4$ **then begin**
{ v moves to C_y }
Remove v from its current clique
$C_y \leftarrow C_y \cup \{v\}$
clique(v) $\leftarrow y$
end
Mark the edge (u, v)
until all edges in E_c are marked

Example.

In this example, we allocate a program consisting of four tasks on a distributed system of two processors. The system tasks and the processing elements are represented using the split graph model shown in Figure 4. The system parameters such as the task execution cost and the communication cost between any pair of tasks are represented in the split graph model as well. For example, the execution cost of running task T_1 on p_1 and p_2 are 10 and 20, respectively. Also the communication cost between T_1 and T_4, if they are assigned to different processing elements, is equal to 50. The step by step execution of the algorithm on this example can be described as follows.

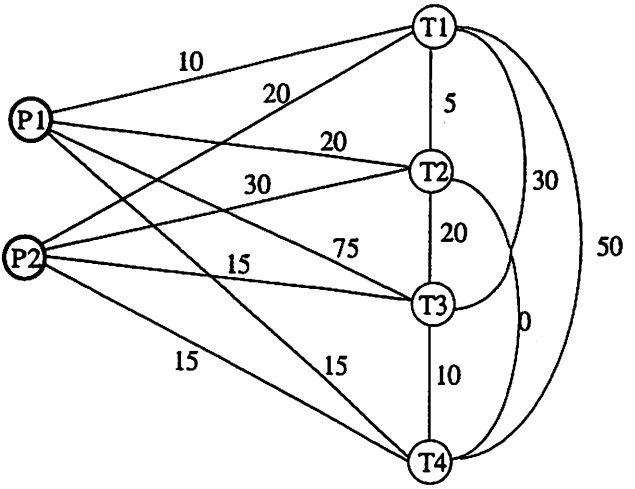

Figure 4. An Example of the Split Graph Model
for 4 Tasks on a 2-Processor System

First, we construct an instance of the clique partitioning problem as follows.

$V_1 = \{p_1, p_2\}$
$V_2 = \{T_1, T_2, T_3, T_4\}$
$\ell(e) = C_{ij}$, for any edge $e = (T_i, T_j)$
$\ell(e) = \beta - X_{ij}$, where $e = (T_i, P_j)$ and $T_i \in V_2, P_j \in V_1$,
β is a large positive integer

Clearly the weights on the complete edges will remain the same, while the weights on the cross edges will change. We choose β to be 200. The new weights on the cross edges are:

$\ell(T_1, p_1) = 190$ $\quad$ $\ell(T_1, p_2) = 180$
$\ell(T_2, p_1) = 180$ $\quad$ $\ell(T_2, p_2) = 170$
$\ell(T_3, p_1) = 125$ $\quad$ $\ell(T_3, p_2) = 185$
$\ell(T_4, p_1) = 185$ $\quad$ $\ell(T_4, p_2) = 185$

After the preprocessing and initialization step, the complete edges are sorted and the situation can be summarized as follows:

$C_1 = \{p_1\}$, $C_2 = \{p_2\}$
$clique(T_1) = 0$, $clique(T_2) = 0$, $clique(T_3) = 0$, $clique(T_4) = 0$.

$$F(T_1,T_2,C_1) = 0 \qquad F(T_1,T_2,C_2) = 20$$
$$F(T_1,T_3,C_1) = 60 \qquad F(T_1,T_3,C_2) = 10$$
$$F(T_1,T_4,C_1) = 0 \qquad F(T_1,T_4,C_2) = 10$$
$$F(T_2,T_3,C_1) = 60 \qquad F(T_2,T_3,C_2) = 10$$
$$F(T_2,T_4,C_1) = 0 \qquad F(T_2,T_4,C_2) = 10$$
$$F(T_3,T_4,C_1) = 60 \qquad F(T_3,T_4,C_2) = 0$$

In the first iteration of the main loop, the edge (T_1,T_4) is considered. The weight of the edge is compared to the values of the cost function for each clique. Since $\ell(T_1,T_4)$ is greater than both $F(T_1,T_4,C_1)$ and $F(T_1,T_4,C_2)$, both nodes T_1 and T_4 are assigned to the same clique; the one with the minimum cost function (C_1). After this iteration, the situation can be summarized as follows:

$$C_1 = \{p_1,T_1,T_4\},\ C_2 = \{p_2\}$$
$$clique(T_1) = 1,\ clique(T_2) = 0,\ clique(T_3) = 0,\ clique(T_4) = 1.$$

In the second iteration, the algorithm considers the edge (T_1,T_3). Again, since $\ell(T_1,T_3)$ is greater than both $F(T_1,T_3,C_1)$ and $F(T_1,T_3,C_2)$, the algorithm initially recommends the assignment of both nodes to clique C_2. Since $clique(T_3)$ initially equals zero, T_3 is assigned to C_2. Since $clique(T_1) = 1$, the total weight when T_1 remains in C_1 is compared to the total weight if T_1 moves to C_2. It turns out that keeping T_1 in C_1 will result in a higher total weight. The situation after the second iteration is:

$$C_1 = \{p_1,T_1,T_4\},\ C_2 = \{p_2,T_3\}$$
$$clique(T_1) = 1,\ clique(T_2) = 0,\ clique(T_3) = 2,\ clique(T_4) = 1.$$

Similarly, in the third iteration, the values of $F(T_2,T_3,C_1)$ and $F(T_2,T_3,C_2)$ are compared with the weight $\ell(T_2,T_3)$. Since, the value of $F(T_2,T_3,C_2)$ is the minimum among the three, the algorithm recommends the assignment of both nodes on C_2. Because of the fact that $clique(T_2) = 0$ and $clique(T_3) = 2$, this recommendation carries. The situation now is:

$$C_1 = \{p_1,t_1,T_4\},\ C_2 = \{p_2,T_3,T_2\}$$
$$clique(T_1) = 1,\ clique(T_2) = 2,\ clique(T_3) = 2,\ clique(T_4) = 1.$$

It can be verified that the last three iterations of the algorithm will maintain the above assignment. Figure 5 shows the task allocation of the original problem on the two processors. Notice that the numbers next to the edges are the original weights in the task allocation problem. In this case the allocation is optimal, when the optimization criterion is to minimize the total execution and communication costs.

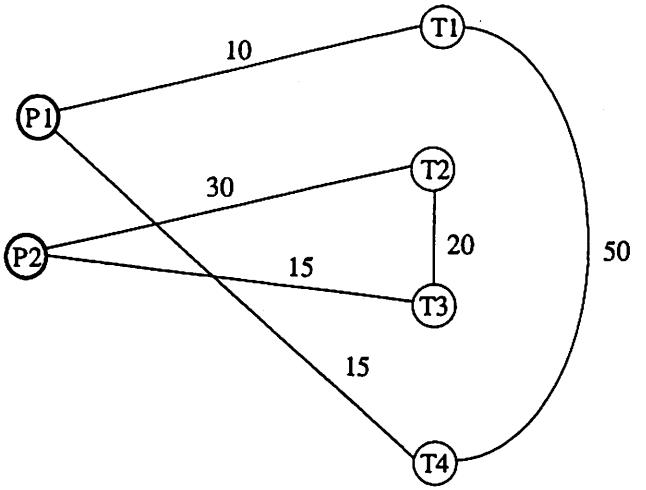

Figure 5. The Task Allocation for Split Graph Model of Figure 4

Lemma 2. *The time complexity of algorithm Partition is of order* $O\left(n^2(m+n)\right)$, *where* n *is the size of the complete set and* m *is the size of the independent set in the split graph.*

Proof: In the preprocessing step, the order of sorting the complete edges is of order $O(n^2 \log n)$, and the order of evaluating the cost function F is $O(n^2 m)$. In the clique partitioning step, the main loop is repeated $n(n-1)/2$ times, that is, the number of complete edges. In each iteration, obtaining p_x and p_y is done in $O(m)$ operations and evaluating the weight of the four cases is of order $O(n)$. This implies that the complexity of the clique partitioning step is $O\left(n^2(m+n)\right)$. Since the complexity of the preprocessing step is $O\left(n^2(m+\log n)\right)$, it follows that the complexity of algorithm Partition is of order $O\left(n^2(m+n)\right)$. ∎

Theorem 2. *Algorithm Partition generates an optimal solution if* $F(u,v,C_i)$ *is* either *equal to zero* or *greater than* $\Sigma\ell(u,v) \forall u,v \in V_2 \& 1 \leq i \leq m$, *and* $\ell(u,p_i) \neq \ell(u,p_j)$, *for any two nodes* $p_i, p_j \in V_1$, $u \in V_2$.

Proof: In this case, the weights on the cross edges dominates the weights on the complete edges in determining the assignment of each node. Recall that $F(u,v,C_i) = 0$ implies that the clique C_i is the best clique to contain the nodes u and v in terms of the weights on the cross edges. Also, $F(u,v,C_i) > \ell(u,v)$ implies that the weight of assigning both nodes to clique C_i is lower than the weight on the complete edge between the two nodes. We prove the theorem by induction on the number of nodes n. If n equals one then algorithm Partition will assign the only node u to the clique C_i such that (u,p_i) is maximum, which will provide an optimal solution. If the algorithm provides an optimal solution for $n-1$ nodes,

we show that the optimality is preserved after assigning the nth node, u. There are two cases to consider. In the first case, $F(u, v, C_i) > \Sigma \ell(u, v) \forall v \in V_2$, and $\forall p_i, p_j \in V_1, 1 \leq i \leq m$. In this case, u will be assigned to clique C_i that contains p_i such that (u, p_i) is maximum. The obtained assignment will have the maximum total weight because if u is assigned to another clique C_j, then the total weight will be decreased by $F(u, v, p_j)$ and increased by at most $\Sigma \ell(u, v)$, for all tasks in C_j. Hence, assigning u to clique C_j will result in a lower total weight. In the second case, if $F(u, v, C_i)$ is equal to zero for some clique C_i, then algorithm Partition will assign node u to clique C_i. Clearly, assigning node u to any clique other than C_i will result in a lower total weight, since $\ell(u, p_i) \neq \ell(u, p_j) \Rightarrow F(u, v, C_j) \neq 0$, for any clique C_j. ∎

The above theorem shows that while the algorithm handles the general case of the partitioning problem, it recognizes this restricted case and provides an optimal partitioning. Although the above case is very restricted, it gives an indication that the proposed heuristic performs better when the weights on the cross edges are relatively high compared to the weights of the complete edges. In other words, using algorithm Partition for solving the task allocation problem, it will perform better when the average execution cost is relatively high compared to the average communication cost. This observation is supported by the experimental results given in Section 6.

6. Experimental results.

In this section, we present the results of a number of simulation experiments designed to study the performance of the algorithm Partition to solve the task allocation problem. The performance of algorithm Partition was compared to the optimal, which was obtained using exhaustive search. The experimental results suggest that the performance of algorithm Partition lies within a factor of 2 of the optimal when the average execution cost is greater than the average communication cost. We define the following two performance measures:

$$M = \frac{\text{total cost of assignment found by Partition}}{\text{total cost of optimal assignment}}$$

$$R = \frac{\text{average communication cost}}{\text{average execution cost}}.$$

We generated 400 complete split graphs with different number of tasks and processing elements. The number of tasks ranged between 4 and 64 while the number of processing elements ranged between 2 and 16. Task execution cost was randomly generated using exponential distribution with an average cost in the range [10, 1000] units of cost. The communication cost between tasks allocated to different processing elements was randomly generated using a uniform distribution with an average cost in the range [0, 1000] units of cost. The experiments were conducted on a Sequent symmetry with 14 processors.

The detailed results of the simulation experiments are reported in [2]. The following table summarizes the relevant experimental results. The entries in the table represent the percentages of the number of simulation runs for each value of M and R. For example, the first two entries in the table indicate that when $R = 20$, in 45% of the runs, algorithm Partition provided an optimal assignment, while in 50% of the runs, it provided an assignment with the ratio between the cost found by Partition and the optimal cost ≤ 1.2.

Table 1

R	$M = 1.0$	≤ 1.2	≤ 1.4	≤ 1.6	≤ 1.8	≤ 2.0
20	45%	50%	50%	75%	80%	85%
10	45%	50%	50%	75%	80%	85%
1	50%	50%	72%	80%	80%	95%
0.5	70%	83%	96%	96%	98%	100%
0.2	100%	100%	100%	100%	100%	100%
0.1	100%	100%	100%	100%	100%	100%

7. Conclusions.

In this paper, we introduced a new graph theoretic approach to model the problem of task allocation on distributed systems. Our model is based on a special class of graphs called split graphs. The split graph model represents the communication cost among tasks as well as the execution cost of each task on the processors. There are several optimization criteria considered when allocating a set of tasks to a distributed system. The objective function of the proposed algorithm is to minimize the total cost which is the summation of the execution cost and the interprocessor communication cost. We showed that the task allocation problem is equivalent to the problem of weighted clique partitioning in complete split graphs and hence is NP-complete. We also devised a new task allocation algorithm based on the problem of weighted clique partitioning in split graphs. We showed that the algorithm generates optimal solutions in some cases, while performing fairly well in general.

Acknowledgment.

We would like to thank Yinghua Huang for her help in writing the programs and conducting the different experiments. We also would like to thank the anonymous reviewer for his/her valuable comments and corrections that have helped us to improve the paper.

References

1. H. Ali and H. El-Rewini, *A proof that task allocation in distributed systems is NP-complete*, Technical Report (October 1992), Department of Mathematics and Computer Science, University of Nebraska at Omaha.
2. H. Ali, H. El-Rewini, and Y. Huang, *Experimental results for task allocation heuristics using split graph model*, Technical Report (April 1992), Department of Mathematics and Computer Science, University of Nebraska at Omaha.
3. S. Bokhari, *A shortest tree algorithm for optimal assignments across space and time in distributed processor system*, IEEE Transaction on Software Engineering **SE-7, no. 6** (November 1981).
4. S. Bokhari Dual processing scheduling with dynamic reassignment, IEEE Transactions on Software Eng. (May 1978), 254–258.
5. T. Chou and J. Abraham, *Load balancing in distributed systems*, IEEE Transaction on Software Engineering **SE-8, no. 4** (July 1981).
6. W. Chu, L. Holloway, M. Lan, and K. Efe, *Task allocation in distributed data processing*, IEEE Computer (November 1980), 57–69.
7. K. Efe, *Heuristic models of task assignment scheduling in distributed systems*, IEEE Computer (June 1982), 50–56.
8. M. Garey and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman and Company, San Fransisco, 1979.
9. M. Golumbic, "Algorithmic Graph Theory and Perfect Graphs", Academic Press, New York, 1980.
10. V. Lo, *Heuristic algorithms for task assignment in distributed systems*, Proc. 4th Int. Conf. Distr. Comput. Syst. (May 1984), 30–39.
11. P. Ma, E.Lee, and M. Tsuchiya, *A task allocation model for distributed computing systems*, IEEE Trans. Comput. (January 1982), 41–47.
12. C. Shen, and W. Tsai, *A graph matching approach to optimal task assignment in distributed computing systems using a minimax critrion*, IEEE Trans. on Comput. (March 1985), 197–203.
13. H, Stone, and S. Bokhari, *Control of distributed processes*, IEEE Computer (July 1987), 85–93.
14. H. Stone, *Multiprocessor scheduling with the aid of network flow algorithms*, IEEE Trans. Software Eng. (January 1977), 85–93.