Optimal Replication of Series-Parallel Graphs for Computation-Intensive Applications*

Sheng-Tzong Cheng
Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland, College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

Abstract

We consider the replication problem of series-parallel (SP) task graphs where each task may run on more than one processor. The objective of the problem is to minimize the total cost of task execution and interprocessor communication. We call it, the *minimum cost replication problem for SP graphs* (MCRP-SP). In this paper, we adopt a new communication model where the purpose of replication is to reduce the total cost. The class of applications we consider is computation-intensive applications in which the execution cost of a task is greater than its communication cost. The complexity of MCRP-SP for such applications is proved to be NP-complete. We present a branch-and-bound method to find an optimal solution as well as an approximation approach for suboptimal solution. The numerical results show that such replication may lead to a lower cost than the optimal assignment problem (in which each task is assigned to only one processor) does. The proposed optimal solution has the complexity of $O(n^2 2^n M)$, while the approximation solution has $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.

1 Introduction

Distributed computer systems have often resulted in improved reliability, flexibility, throughput, fault tolerance and resource sharing. In order to use the processors available in a distributed system, the tasks have to be allocated to the processors. The allocation problem is one of the basic problems of distributed computing whose solution has a far reaching impact on the usability and efficiency of a distributed system. Clearly, the tasks of an application have to be executed satisfying the precedence and other synchronization constraints among them. (Such constraints are often specified in the form of a task graph.)

In executing an application, defined by its task graph, we have the option of restricting ourselves to having only one copy of each task. The allocation problem, in this case, is referred to as *assignment problem*. If, on the other hand, a task may be replicated multiple times, the general problem is called the *replication problem*. In this paper, we consider the replication problem and present an algorithm to find the optimal replication of series-parallel graphs for computation-intensive applications.

For distributed processing applications, the objective of the allocation problem may be the minimum completion time, processor load balancing, or total cost of execution and communication, etc. For the assignment problem where the objective is to minimize the total cost of execution and interprocessor communication, Stone [11] and Towsley [12] presented $O(n^3M)$ algorithms for tree-structure and series-parallel graphs, respectively, of M tasks and n processors. For general task graphs, the assignment problem has been proven [9] to be NP-complete. Many papers [8][9][10] presented branch-and-bound methods which yielded an optimal result. Other heuristic methods have been considered by Lo [7] and Price and Krishnaprasad [5]. All these works focused on the assignment problem.

Traditionally, the main purpose of replicating a task on multiple processors is to increase the degree of fault tolerance [2][6]. If some processors in the distributed system fail, the application may still survive using other copies. In such a communication model, a task has to communicate with multiple copies of other tasks. As a consequence, the total cost of execution and communication of the replication problem will be bigger than that of the assignment problem. In this paper, we adopt another communication model in which the replication of a task is not for the sake of fault tolerance but for decreasing of the total cost. In our model, each task may have more than one copy and it may start its execution after receiving necessary data from one copy of each preceding task. Clearly, in a heterogeneous environment the cost of execution of a task depends on the processor on which it executes, and the communication costs depend on the topology, communication medium, protocols used, etc. When a task t is allowed to have only one copy in the system, the sum of the interprocessor communication costs between t and other tasks may be large. Sometimes it will be more beneficial if we replicate t onto multiple processors to reduce the inter-processor communication, and to fully utilize the available processors in the systems. Such replication may lead to a lower total cost than the optimal assignment problem does. An example illustrating this point is presented in Section 3.

In the assignment problem, polynomial-time algorithms exist for special cases, such as tree-structure [11] and series-parallel [12] task graphs. This paper represents one of the first few attempts at finding special cases for the replication problem. The class of applications we consider in this paper is computation-intensive applications in which the execution cost of a task is greater than its communication cost. Such applications can be found in an enormous number of fields, such as digital signal processing, weather forecasting, game searching, etc. We formally define a computation-intensive application in Section 2.2. In this paper, we prove that for the computation-intensive applications, the replication problem is NP-complete, and we present a branch-and-bound algorithm to solve it. The worst-case complexity of the solution is $O(n^2 2^n M)$. Note that the algorithm is able to solve the problem in the complexity of the linear function of M.

We also develop an approximation approach to solve the problem in polynomial time. Given a forker task s with K successors in the SP graph, the method tries to allocate s to processors based on iterative selection. The complexity of the iterative selection for a forker is $O(n^2 K^2)$, while the overall solution for an SP graph is $O(n^4 M^2)$.

In the remainder of this paper, the series-parallel graph model and the computation model are described in section 2. In section 3, the replication problem is formulated as the minimum cost 0-1 integer programming problem and the proof of NP completeness is given. A branch-and-bound algorithm and numerical results are given in section 4, while the approximation methods and results are given in section 5. The overall algorithm is presented and conclusion remark is drawn in section 6.

2 Definitions

2.1 Graph Model

A *series-parallel* (SP) graph, $G = (V, E)$, is a directed graph of type p, where $p \in \{T_{unit},\ T_{chain},\ T_{and},\ T_{or}\}$ and G has a source node (of indegree 0) and a sink node (of outdegree 0). An SP graph can be constructed by applying the following rules recursively.

1. A graph $G = (V, E) = (\{v\}, \phi)$ is an SP graph of type T_{unit}. (Node v is the source and the sink of G.)

2. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are SP graphs then $G' = (V', E')$ is an SP graph of type T_{chain}, where $V' = V_1 \cup V_2$ and $E' = E_1 \cup E_2 \cup \{<$sink of G_1, source of $G_2>\}$.

3. If each graph $G_i = (V_i,E_i)$ with source-sink pair (s_i,t_i), where s_i is of outdegree 1, is an SP graph, $\forall\ i = 1,2,\ldots,n$, and new nodes $s' \notin V_i$ and $t' \notin V_i$, $\forall\ i$ are given then $G' = (V', E')$ is an SP graph of type T_{and}(or type T_{or}), where $V' = V_1 \cup V_2 \cup \ldots \cup V_n \cup \{s', t'\}$ and $E' = E_1 \cup E_2 \cup \ldots \cup E_n \cup \{< s', s_i > \mid \forall\ i = 1,2,\ldots,n\ \} \cup \{< t_i, t' > \mid \forall\ i = 1,2,\ldots,n\ \}$. The source of G', s', is called the *forker* of G'. The sink of G', t', is called the *joiner* of G'. G' is an SP graph of type T_{and}(or type T_{or}) if there exists a *parallel-and* (or *parallel-or*) relation among G_i's.

A convenient way of representing the structure of an SP graph is via a parsing tree [4]. The transformation of an SP graph to a parsing tree can be done in a recursive way. There are four kinds of internal nodes in a parsing tree: T_{unit}, T_{chain}, T_{and} and T_{or} nodes. A T_{unit} node has only one child, while a T_{chain} node has more than one child. Every internal node x, along with all its descendant nodes induces a subtree S_x which describes an SP subgraph G_x of G. Each leaf node in S_x corresponds to an SP graph of type T_{unit}. A T_{and}(or T_{or}) node y consists of its type T_{and}(or T_{or}) along with the forker and joiner nodes of G_y. We give an example of an SP graph G, and its parsing tree $T(G)$ in Figure 1.

2.2 Computational Model

An application program consists of M tasks labeled $m = 1, 2, \ldots, M$. Its behavior is represented by an SP graph with the tasks correspond to the nodes. Each task may be replicated onto more than one processor. A *task instance* $t_{i,p}$ is a replication of task i on processor p. A directed edge $< i, j >$ between nodes i and j exists if the execution of task j follows that of task i. Associated with each edge $< i, j >$ is the communication cost incurred by the application. We are concerned with types of applications where the cost of execution of a task is always greater than the communication overhead it needs. The model is stated as follows.

Given a distributed system S with n processors connected by a communication network, an application is *computation-intensive* if its associated SP graph $G = (V, E)$ on S satisfies the following conditions:

1. $\mu_{i,j}(p, q) \geq 0$,

2. $\sum_{q=1}^{n} \mu_{i,j}(p, q) \leq \min_p(e_{i,p})$, $\forall < i, j > \in E$, and $1 \leq p \leq n$, where

- $\mu_{i,j}(p, q)$ is the communication cost between tasks i and j when they are assigned to processors p and q respectively, and

- $e_{i,p}$ is the execution cost when task i is assigned to processor p.

The first condition states that the communication cost between any two task instances (e.g. $t_{i,p}$ and $t_{j,q}$) is not negative. The second one depicts that for every edge $< i, j >$, the worst-case communication cost between any task instance $t_{i,p}$ and all its successor task instances (i.e. $t_{j,q}$'s, $\forall q$) is less than the minimum execution cost of task i.

2.3 Communication Model

The communication model we considered is different from that of reliability-oriented replication. In reliability-oriented replication problem, the objective is to increase the degree of fault tolerance. To detect fault and maintain data consistency, each task has to receive multiple copies of data from several task instances if its predecessor is replicated in more than one place.

The purpose of the replication problem considered in this paper is to decrease the sum of execution and communication costs. Under such consideration, there is no need to enforce plural communication between any two task instances. Hence, we propose the *1-out-of-n* communication model. In the model, for each edge $< i, j > \in E$, a task instance $t_{j,q}$ may start its execution if it receives the data from *any one* task instance of its predecessor, task i.

3 Problem Formulation and Complexity

Based on the computational model presented in Section 2.2, the problem of minimizing the total sum of execution and communication costs for an SP task graph can be approached by replication of tasks. An example where the replication may lead to a lower sum of execution costs and communication costs is given in Figure 2, where the number of processors in the system is two, and the execution costs and communication costs are listed in e table and μ table respectively. If each task is allowed to run on at most one processor, then the optimal allocation will be to assign task a to processor 1, b to 1, c to 1, d to 2, e to 2, and f to 1. The minimum cost is 68. However, if each task is allowed to be replicated more than one copies, (i.e. to replicate task a to processors 1 and 2), then the cost is 67.

We introduce integer variable $X_{i,p}$'s, $\forall\ 1 \leq i \leq M$ and $1 \leq p \leq n$, to formulate the problem where each $X_{i,p} = 1$ if task i is replicated on processor p; and $= 0$, otherwise. We define a binary function $\delta(x)$. If $x > 0$ then $\delta(x) = 1$ else $\delta(x) = 0$. We also associate an *allocated flag* $F(w)$ with each node w in the parsing tree, where $F(w) = 1$ if the allocation for tasks in the subtree S_w is valid; and $= 0$, otherwise. A valid allocation for the tasks in S_w is an allocation that follows the semantics of T_{chain}, T_{and}, and T_{or} subgraphs. A valid allocation is not necessarily the allocation in which each task in S_w is allocated to at least one processor. Some tasks in T_{or} subgraphs may be neglected without effecting the successful execution of an SP graph.

Given an SP graph G, its parsing tree $T(G)$ and any internal node w in $T(G)$, allocated flag $F(w)$ can be recursively computed:

1. if w is a T_{unit} node with a child i, then

$$F(w) = F(i) = \delta(\sum_{p=1}^{n} X_{i,p})$$

2. if w is a T_{chain} node with c children, $F(w) = F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

3. if w is a T_{and} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

4. if w is a T_{or} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times \delta(F(child_1) + F(child_2) + \ldots + F(child_c))$.

The minimum cost replication problem for SP graphs, MCRP-SP, can be formulated as 0-1 integer programming problem, i.e:

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,\ 1\leq q\leq n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q})\ ]$$

$$\text{subject to } F(r) = 1, \text{ where } r \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i, p. \tag{1}$$

The restricted problem which allows each task to run on at most one processor has the following formulation.

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,p,q} \mu_{i,j} * X_{i,p} * X_{j,q}\ ]$$

$$\text{subject to } \sum_{p=1}^{n} X_{i,p} \leq 1 \text{ and } F(r) = 1,$$

$$\text{where } r \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i, p. \tag{2}$$

The task assignment problem (2) for SP graphs of M tasks onto n processors, has been solved in $O(n^3 M)$ time [12]. However,the multiprocessor task assignment for general types of task graphs without replication has been reported to be NP-complete [9]. As for the MCRP-SP problem, it can be shown to be NP-complete. In this paper, we are able to solve the problem and present a linear-time algorithm that is linear in the number of tasks when the number of processors is fixed for computation-intensive applications.

3.1 Assignment Graph

Bokhari [1] introduced the assignment graph to solve the task assignment problem (2). To prove the NP completeness of problem (1) and solve the problem, we also adopt the concept of the assignment graph of an SP graph. The assignment graph of an SP graph can be defined similarly. The following definitions apply to the assignment graph. And we draw up an assignment graph for an SP graph in Figure 3.

1. It is a directed graph with weighted nodes and edges.
2. It has $M \times n$ nodes. Each weighted node is labeled with a task instance, $t_{i,p}$.
3. A *layer* i is the collection of n weighted nodes ($t_{i,1}$, $t_{i,2}$, ..., and $t_{i,n}$). Each layer of the graph corresponds to a node in the SP graph. The layer corresponding to the source (sink) is called source (sink) layer.
4. A part of the assignment graph corresponds to an SP subgraph of type T_{chain}, T_{and} or T_{or} is called a T_{chain}, T_{and} or T_{or} *limb* respectively.
5. Communication costs are accounted for by giving the weight $\mu_{i,j}(p,q)$ to the edge going from $t_{i,p}$ to $t_{j,q}$.
6. Execution costs are assigned to the corresponding weighted nodes.

Given an assignment graph, Bokhari [1] solves Problem (2) by selecting one weighted node from each layer and including the weighted edges between any two selected nodes. This resulting subgraph is called an *allocation graph*. To solve Problem (1), more than one weighted node from each layer may be chosen. Similarly, a *replication graph* for Problem (1) can be constructed from an assignment graph by including all selected nodes and edges between these nodes. Examples of an allocation graph and a replication graph are shown in Figure 4 for an assignment graph shown in Figure 3. Note that for each node x in the replication graph there is only one edge incident to it from each predecessor layer of x.

In a replication graph, each layer may have more than one selected node. Let Variable $\bar{X}_l$ = ($X_{l,1}$, $X_{l,2}$, ..., $X_{l,n}$) be a replication vector for layer l in a replication graph. We define the minimum *activation cost* of vector $\bar{X}_i$ for layer i , $A_i(\bar{X}_i)$, to be the minimum sum of the weights of all possible nodes and edges leading to the selected nodes of layer i in a replication graph. Then the goal of Problem (1) can be achieved by computing the minimal value of $\{A_{\text{sink}}(\bar{X}_{\text{sink}}) + \sum_{p=1}^{n} X_{\text{sink},p} * e_{\text{sink},p}\}$ over all possible values of $\bar{X}_{\text{sink}}$.

3.2 Complexity

In this section, we can show that Problem (1) for a computation-intensive application is NP-complete provided we prove the following:

Lemma 1: For any layer l in the replication graph, the minimum activation cost for two selected nodes $t_{l,p}$ and $t_{l,q}$ will be always greater than that for either node $t_{l,p}$ or $t_{l,q}$ only.

Proof: The Lemma can be proven by contradiction. Let A_1 be the the minimum activation cost for two nodes $t_{l,p}$ and $t_{l,q}$, and A_2 and A_3 be the minimum costs for $t_{l,p}$ and $t_{l,q}$ respectively. Assume that $A_1 < A_2$ and $A_1 < A_3$. Since A_1 includes the activation cost of node $t_{l,p}$, an activation cost for $t_{l,p}$ only can be obtained from A_1. The obtained value c is not necessarily the minimum value for $t_{l,p}$, hence $A_2 \leq c$. The value c is obtained by removing some weighted nodes and edges from replication graph. This implies that $c < A_1$. From above, we find that $A_2 < A_1$, which contradicts the assumption. The same reasoning can be applied to A_3 and reaches a contradiction. Therefore, the assumptions are incorrect and Lemma 1 holds.

□

Lemma 1 can be further extended to the cases where more than two weighted nodes are chosen. The conclusion we can draw is that the more nodes are selected from a layer, the bigger the activation cost is.

Lemma 2: Given a computation-intensive application with its SP task graph $G = (V, E)$ and its assignment graph, if node i has outdegree one and edge $< i, j > \in E$, then for any vector $\bar{X}_j$, the minimal activation cost $A_j(\bar{X}_j)$ can be obtained by choosing only one weighted node from layer i. (i.e. $\sum_{p=1}^{n} X_{i,p} = 1$)

Proof: The Lemma can be proven by contradiction. Since node i has outdegree one and edge

$< i,j > \in E$, we know that

$$A_j(\bar{X}_j) = \min_{\bar{X}_i}\{A_i(\bar{X}_i) + \sum_{p=1}^{n} X_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))\}.$$

Let us assume that the above equation reaches a minimal value m when more than one node from layer i is selected and the optimal replication vector is $\bar{X}_i^0$. Since $\sum_{p=1}^{n} X_{i,p} > 1$ for $\bar{X}_i^0$, we may remove one selected node from layer i and obtain a new vector $\bar{X}_i'$. Without loss of generality, let us remove $t_{i,r}$. By removing node $t_{i,r}$, a new value m' is obtained. Since m is the minimum value for layer i, it implies that $m \leq m'$.

From Lemma 1, we obtain that $A_i(\bar{X}_i') < A_i(\bar{X}_i^0)$. And for a computation-intensive application, the following holds that $\sum_{q=1}^{n} \mu_{i,j}(p,q) \leq \min_p(e_{i,p})$, $\forall\ 1 \leq p \leq n$. Then,

$$\begin{aligned}
m' &= A_i(\bar{X}_i') + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&< A_i(\bar{X}_i^0) + (\sum_{p=1}^{n} X^0_{i,p} * e_{i,p} \ - \ e_{i,r}) + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&= A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - e_{i,r} \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - \min_p(e_{i,p}) \\
&\leq A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - \sum_{q=1}^{n} \mu_{i,j}(p,q) \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p}
\end{aligned}$$

$$< \quad A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X_{i,p}^0 * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}^0 = 1} (X_{j,q} * \mu_{i,j}(p,q)) = m.$$

The result, $m' < m$, contradicts our assumption. It means that the assumption is wrong and Lemma 2 holds.

□

Lemma 3: Given a computation-intensive application with its SP task graph G, the objective of the minimum cost can be achieved by considering only the replication of the forkers.

Proof: We proceed to prove the lemma by contradiction. Let the minimum cost for task replication problem be z_0 if only the forkers(i.e. outdegree > 1) are allowed to run on more than one processor. Assume the total cost can be reduced further by replicating some task i which is not a forker. Then there are two possible cases for i:

1. i has outdegree 0.
2. i has outdegree 1.

In case 1, i is the sink of the whole graph. Also i may be the joiner of some SP subgraphs. If i is allowed to run on an extra processor b, which is different from the one which i is initially assigned to (when z_0 is obtained), then the new cost will be $z_0 + e_{i,b} + \sum_{<d,i>\in E} \mu_{d,i}$. Apparently, the new cost is greater than z_0. This contradicts our assumption that the total cost can be reduced further by replicating task i.

In case 2, i has one successor. Let $< i, j > \in E$. From the assumption, we know that the replication of i can reduce the total cost. Hence, the minimum activation cost for task instances in layer j, $A_j(\bar{X}_j)$, is obtained when task i is replicated onto more than one processor. This contradicts Lemma 2. Hence, the assumption is incorrect and the objective of the minimum cost can be achieved by considering only the replication of the forkers.

□

Lemma 3 tells that, given an SP graph, if we can find out the optimal replication for the forkers, Problem (1) for computation-intensive applications can be solved. Now, we show that the problem of finding an optimal replication for the forkers in an SP graph is NP-complete. First, a special form of the replication problem is introduced.

Uni-Cost Task Replication (UCTR) problem is stated as follows:

INSTANCE: Graph $G' = (V', E')$, $V' = V_1' \cup V_2'$, where $\mid V_1' \mid = n$ and $\mid V_2' \mid = m$. If $x \in V_1'$ and $y \in V_2'$ then edge $< x, y > \in E'$ (i.e. $\mid E' \mid = m \times n$). For each $x \in V_1'$, there is an activation cost m. Associated with each edge $< x, y > \in E'$, there is a communication cost $d_{x,y} = n \times m$ or 0. A positive integer $K \leq n \times m$ is also given.

QUESTION: Is there a feasible subset $V_k \subseteq V_1'$ such that, we have

$$[\sum_{x \in V_k} m + \sum_{y \in V_2'} \min_{x \in V_k} (d_{x,y})] \leq K? \tag{3}$$

[Theorem 1]: Uni-Cost Task Replication problem is NP-Complete.

[**Proof**]: The problem is in NP because a subset V_k, if it exists, can be checked to see if the sum of activation costs and communication costs is less than or equal to K. We shall now transform the VERTEX COVER [3] problem to this problem. Given any graph $G = (V,E)$ and an integer $C \leq \mid V \mid$, we shall construct a new graph $G' = (V',E')$ and $V' = V_1' \cup V_2'$, such that there exists a VERTEX COVER of size C or less in G if and only if there is a feasible subset of V_1' in G'. Let $\mid V \mid = n$ and $\mid E \mid = m$. To construct G', (1) we create a vertex v_i for each node in V, (2) we number the edges in E, and (3) we create a vertex b_j for each edge $< u,v > \in E$ where u, $v \in V$. We define $K = m \times C$, $V_1' = \{v_1, v_2, \ldots, v_n\}$, $V_2' = \{b_1, b_2, \ldots, b_m\}$ and $E' = \{< v_x,b_y > \mid v_x \in V_1', b_y \in V_2' \}$. Let $d_{v_x,b_y} = 0$, if v_x is an end point of the corresponding edge of vertex b_y; and $= n \times m$, otherwise. An illustration, where $n = 7$ and $m = 9$, is shown in Figure 5.

Let us now argue that there exists a vertex cover of size C or less in G if and only if there is a feasible subset of V_1' in G' to satisfy that the sum of activation cost and communication cost is $m \times C$ or less. Suppose there is a vertex cover of size C, then for each vertex b_y ($= < u,v >$) in V_2', at least one of u and v belongs to the vertex cover. By selecting all the vertices in the vertex cover into the subset of V_1', we know that the sum in Eq. (3) will be $m \times C$. Since $C \leq n$, it implies that $m \times C \leq n \times m$.

Conversely, for any feasible subset $V_k \subseteq V_1'$ such that the total cost is equal to or less than

mC, we can see that the second term of Eq. (3) (i.e. the sum of communication cost) must be zero. Suppose, for some $g_y \in V_2'$, the minimum communication cost between g_y and vertices in V' is nonzero, then the communication cost will be at least $m \times n$. Since $C \leq n$, it implies that $m \times n \geq m \times C$. The total cost in Eq. (3) will be greater than $m \times C$, which is a contradiction. Thus the minimum communication cost between any vertex in V_2' and any vertex in V_k is zero. It means that at least one of two end points of each edge in E belongs to V_k. Since, there is at most C vertices in V_k (the activation cost for each vertex is m), and by selecting the vertices in V_k, we obtain a vertex cover of size C or less in G.

□

[**Theorem 2**]: *The problem,* MCRP-SP *for computation-intensive applications, is NP-complete.*

[**Proof**]: From Lemma 3, we know that only the forker in an SP graph of type T_{and} needs to run on more than one processor. Consider the following recognition version of Problem (1) for SP graphs of type T_{and}:

Given a distributed system of n processors, an SP graph $G^a = (V^a, E^a)$ of type T_{and}, its assignment graph H and two positive integers m and r. Let r be a multiple of m, $V^a = \{s, t, 1,2,\ldots,r\}$ and $E^a = \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. Task s (t) is the forker (joiner) of G^a. Execution cost $e_{i,p}$ and communication cost $\mu_{i,j}(p,q)$ are defined in H, $\forall < i,j > \in E^a$ and $\forall\ 1 \leq p,q \leq n$. Integer variable $X_{i,p} = 1$ if task i is assigned to processor p; and $= 0$, otherwise. When a positive integer $K \leq r$ is given, is there an assignment of $X_{i,p}$'s, such that

$$[\ \sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,\ 1\leq q\leq n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q})\ ] \leq K?$$

$$\text{where } \sum_{i,p} X_{i,p} = 1,\ \forall i \neq s, \text{ and } \sum_{i,p} X_{i,p} \geq 1, \text{ if } i = s. \tag{4}$$

We shall transform the UCTR problem to this problem. Given any graph $G' = (V_1' \cup V_2', E')$ considered in UCTR problem, we construct an SP graph of type T_{and}, $G^a = (V^a, E^a)$, and its assignment graph H, such that G' has a feasible subset of V_1' to allow the sum in Eq. (3) is K or less if and only if there is an assignment of $X_{i,p}$'s for G^a and H to satisfy Eq. (4). Let $\mid V_1' \mid = n$,

$| V_2' | = m$, then the unit cost $f = n \times m$. Assign $r = m \times f$ $(= n \times m^2)$ and $K = n \times m$. The forker and joiner of G^a are s and t respectively. Then $V^a = \{s, t, 1, 2, \ldots, r\}$ and $E^a = \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. We assign the execution costs and communication costs in H as follows. An illustration, where $m = 2$ and $n = 3$, is shown in Figure 7.

- $\forall\ 1 \leq p \leq n$, $e_{s,p} = m$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, if $p = 1$ then $e_{i,p} = 0$ else $e_{i,p} = r$.
- $\forall\ 1 \leq p \leq n$, if $p = 1$ then $e_{t,p} = 0$ else $e_{t,p} = r$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, let $q = (i-1)$ div $(m \times n)$, where div is the integral division. If $d_{v_p,b_{q+1}} \neq 0$ then $\mu_{s,i}(p,1) = 1$ else $\mu_{s,i}(p,1) = 0$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, $\forall\ q \neq 1$, $\mu_{s,i}(p,q) = 0$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p,q \leq n$, $\mu_{i,t}(p,q) = 0$.

It is easy to verify that the SP graph constructed by the the above rules is of type T_{and} and computation-intensive. For each node in V_2' of G', we create f nodes in G^a, where the communication cost between each node and source s is either one or zero.

Let us now argue that there exists a feasible subset of V_1' for UCTR problem if and only if there exists a valid assignment of $X_{i,p}$'s such that the total sum in Eq. (4) is K or less. Suppose a feasible subset V_k of V_1' exists such that the sum in Eq. (3) is C $(\leq K)$. Let V_1' be $\{v_1, v_2, \ldots, v_n\}$ Then we can obtain a valid assignment by letting $X_{i,1} = 1$, $X_{i,2} = 0$, …, $X_{i,n} = 0$, $\forall\ 1 \leq i \leq r$, and $X_{t,1} = 1$, $X_{t,2} = 0$, …, $X_{t,n} = 0$, and $X_{s,p} = 1$, if $v_p \in V_k$; and $X_{s,p} = 0$, if $v_p \notin V_k$, $\forall\ 1 \leq p \leq n$. Since each node x in V_2' corresponds to f nodes in G^a, it is sure that the communication cost between node x and any node (v_p) in V_1' is equal to the total communication costs between these f nodes and any task instance of source $(t_{s,p})$ in G^a. By summing up all the costs, we can obtain that the total sum is C. Since $C \leq K \leq n \times m < r$, this is a valid assignment.

Conversely, if there exists an assignment of $X_{i,p}$'s such that the sum in Eq. (4) is K or less, then the following must be true that $X_{i,1} = 1$, $X_{i,2} = 0$, …, $X_{i,n} = 0$, $\forall\ 1 \leq i \leq r$, and $X_{t,1} = 1$, $X_{t,2} = 0$, …, $X_{t,n} = 0$. It is because for some $p \neq 1$, if $X_{i,p} = 1$ then the sum must be greater than

r, which causes a conflict. Hence the second term in Eq. (4) must be zero. Thus, we may obtain a subset of V_1 for UCTR problem by selecting node $x \in V_1$ if $X_{s,x}$ equals 1. Since the first term in Eq. (3) is equivalent to the first term in Eq. (4), the total sum for UCTR problem will be also K or less then.

□

4 Optimal Replication for SP Graphs of Type T_{and}

In this section, we develop the branch-and-bound algorithm to find an optimal solution for T_{and} subgraphs. The non-forker nodes only need to run on one processor. Hence, an optimal assignment of non-forker nodes can be done after an optimal replication for forkers is obtained.

4.1 A Branch-and-Bound Method for Optimal Replication

Consider a T_{and} SP graph with forker-joiner pair (s,h) shown in Figure 6. There are B subgraphs connected by s and h. These B subgraphs have a parallel-and relationship. Since the joiner h has only one copy in optimal solution (i.e. $\sum_{p=1}^{n} X_{h,p} = 1$), we decompose the minimum cost replication problem $\mathcal{P}$ for a T_{and} SP graph into n subproblems $\mathcal{P}^q$, $q = 1, 2, \ldots, n$, where $\mathcal{P}^q$ is to find the minimum cost when the joiner is assigned to processor q (i.e. $X_{h,q} = 1$).

Given a joiner instance $t_{h,q}$, subgraphs G_b's, $b = 1, 2, \ldots, B$, and the minimum costs $C^b_{p,q}$'s between each forker instance $t_{s,p}$ and joiner instance $t_{h,q}$, $\forall\ 1 \leq p \leq n$ and $1 \leq b \leq B$. we further decompose problem $\mathcal{P}^q$ into n subproblems $\mathcal{P}^q_k$, $k = 1, 2, \ldots, n$, where k is the number of replicated copies that the forker s has. Basically, $\mathcal{P}^q_k$ means the problem of finding an optimal replication for k copies of forker s where the joiner h is assigned to processor q. Since the problem of finding an optimal replication for forker s is NP-complete, we propose a branch-and-bound algorithm for each subproblem $\mathcal{P}^q_k$.

We sort the forker instances according to their execution costs $e_{s,p}$'s into non-decreasing order. Without loss of generality, we assume $e_{s,1} \leq e_{s,2} \leq \ldots \leq e_{s,n}$. We represent all the possible combinations that s may be replicated by a combination tree with $\binom{n}{k}$ leaf nodes. To make the solution efficient, we shall not consider all combinations since it is time-consuming. We apply a least-cost branch-and-bound algorithm to find an optimal solution by traversing a small portion of the combination tree.

During the search, we maintain a variable $\hat{z}$ to record the minimum value known so far. The search is done by the expansion of intermediate nodes. Each intermediate node v at level y represents a combination of y out of n forker instances. The expansion of node v generates at most $n-y$ child nodes, while each child node inherits y forker instances from v and adds one distinct forker instance to itself. For example, if node v is represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$, then $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y}, t_{s,i_y+j} \succ$ represents a possible child node of v, $\forall\ 1 \leq j \leq n-i_y$. A combination tree, where $k = 4$ and $n = 6$, is shown in Figure 8. At any intermediate node of a combination tree, we apply an estimation function to compute the least cost this node can achieve. If the estimated cost is greater than $\hat{z}$, then we prune the node and the further expansion of the node is not necessary. Otherwise, we insert this node along with its estimated cost into a queue. The nodes in the queue are sorted into non-decreasing order of their estimated costs, where the first node of the queue is always the next one to be expanded. When the expansion reaches a leaf node, the actual cost of this leaf is computed. If the cost is less than $\hat{z}$, we update $\hat{z}$. The algorithm terminates when the queue is empty.

4.1.1 The Estimation Function

The proposed branch-and-bound algorithm is characterized by the estimation function. Let node v be at level y of the combination tree associated with subproblem $\mathcal{P}_k^q$ and be represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$. Any leaf node that can be reached from node v needs $k-y$ more forker instances. Let $\ell = \prec j_1, j_2, \ldots, j_{k-y} \succ$ be a tuple of $k-y$ instances chosen from the remaining $n-i_y$ instances, where $j_1 < j_2 < \ldots < j_{k-y}$. Let L be the set of all possible ℓ's. Let $g(v)$ be the smallest cost among all leaf nodes that can be reached from node v.

$$g(v) = \sum_{a=1}^{y} e_{s,i_a} \ + \ \min_{\ell \in L}[\sum_{j_x \in \ell} e_{s,j_x} \ + \ \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y \text{ or } p \in \ell} (C_{p,q}^b)\ ] \ + \ e_{h,q}.$$

Since the complexity involved in computing $g(v)$ is $\binom{n-i_y}{k-y}$, we use the following estimation function $est(v)$ to approximate $g(v)$:

$$est(v) = \sum_{a=1}^{y} e_{s,i_a} \; + \sum_{j=i_y+1}^{i_y+k-y} e_{s,j} \; + \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y,i_y+1,i_y+2,\ldots,n} (C^b_{p,q}) \; + \; e_{h,q}. \tag{5}$$

Since

$$\sum_{j=i_y+1}^{i_y+k-y} e_{s,j} \leq \sum_{j_x \in \ell} e_{s,j_x} \quad \text{and} \quad \sum_{b=1}^{B} \min_{p=i_y+1,i_y+2,\ldots,n} (C^b_{p,q}) \leq \sum_{b=1}^{B} \min_{p \in \ell} (C^b_{p,q}) \; ,$$

it is easy to see that $est(v) \leq g(v)$. Hence, we use $est(v)$ as the lower bound of the objective function at node v.

4.1.2 The Proposed Algorithm

Three parameters of the branch-and-bound algorithm are joiner instance ($t_{h,q}$), the number of processors that forker s is allowed to run (k), and the up-to-date minimum cost ($\hat{z}$). The algorithm $BB(k, q, \hat{z})$ is shown in Table 1.

The MCRP-SP problem can be solved by invoking $BB(k, q, \hat{z})$ n^2 times with parameters set to different values. $BB(k, q, \hat{z})$ solves the problem $\mathcal{P}^q_k$, while the whole procedure, shown in Table 2, solves $\mathcal{P}$.

4.2 Performance Evaluation

The essence of the branch-and-bound algorithm is the expansion of the intermediate nodes. Upon the removal of a node from the queue its children are generated and their estimated values are computed. If the estimation function performs well and gives a tight lower bound of objective function, the number of expanded nodes should be small. Then an optimal solution can be found out as soon as possible.

We conduct two sets of experiments to evaluate the performance of the proposed solution. The performance indices we consider are the number of enqueued intermediate nodes (EIM) and the number of visited leaf nodes (VLF) during the search. We calculate EIM and VLF by inserting one counter for each index at lines 13 and 8 of Table 1 respectively. Each time the execution reaches line 13 (8), EIM (VLF) is incremented by 1.

The first set of experiments is on SP graphs of type T_{and} where the communication cost between any two task instances is arbitrary and is generated by random number generator within the range [1,50]. The execution cost for each task instance is also randomly generated within the same range. The second set of experiments is on SP graphs of type T_{and} with the constrain of computation-intensive applications. We vary the size of the problem by assigning different values to the number of processors in the system (n) and the number of parallel-and subgraphs connected by forker and joiner (B). For each size of the problem (n, B), we randomly generate 50 problem instances and solve them. The results, including the average values of EIM and VLF over the solutions of 50 problem instances, are summarized in Table 3.

From Table 3, we find out that the proposed method significantly reduces the number of expansions for intermediate nodes and leaf nodes. For example, for problem size (n, B) = (20, 40), the total number of leaf nodes is 2^{20} (= 1,048,576) if an exhaustive search is applied. However, our algorithm only generates 16,857 nodes on the average, because we apply $est(v)$, $\hat{z}$, and the branch-and-bound approach.

The branch-and-bound approach and the estimation function even perform better for the computation-intensive applications. We can see that EIM and VLF values are much more smaller in Set II than those in Set I. It is because that in the computation-intensive applications an optimal number of replications for the forker is smaller than that in general applications. The $\hat{z}$ value in function $OPT()$ is able to reflect this fact and avoid the unnecessary expansions.

5 Sub-Optimal Replication for SP Graphs of Type T_{and}

The branch-and-bound algorithm in section 4.1 yields an optimal solution for T_{and} subgraphs. However, the complexity involved is in exponential time in the worst case. Hence, we also consider to find a near-optimal solution in polynomial time.

5.1 Approximation Method

For the problem $\mathcal{P}_k^q$ defined in section 4.1, we exploit an approximation approach to solve it in polynomial time. The approach is based on iterative selection in a dynamic programming fashion. Given a joiner instance $t_{h,q}$ and subgraphs G_b, $b = 1, 2, \ldots, B$, and minimum costs $C_{p,q}^b$ between $t_{h,q}$ and $t_{s,p}$, $p = 1, 2, \ldots, n$, and $b = 1, 2, \ldots, B$. we define $Sub(p, b)$ to be the sub-optimal solution for replication of forker s where forker instances $t_{s,1}$, $t_{s,2}$, $\ldots$, $t_{s,p}$ and subgraphs G_1, G_2, $\ldots$, G_b are taken into consideration.

Strategy 1:

$Sub(p, b)$ can be obtained from $Sub(p-1, b)$ by considering one more forker instance $t_{s,p}$. Strategy 1 consists of two steps. The first step is to initialize $Sub(p, b)$ to be $Sub(p-1, b)$ and to determine if $t_{s,p}$ is to be included into $Sub(p, b)$ or not. If yes, then add $t_{s,p}$ in. The second step is to examine if any instances in $Sub(p-1, b)$ should be removed or not. Due to the possible inclusion of $t_{s,p}$ in the first step, we may obtain a lower cost if we remove some instances $t_{s,i}$'s, $i < p$, and reassign the communications for some graphs G_j's from $t_{s,i}$'s to $t_{s,p}$.

Strategy 2:

$Sub(p, b)$ can also be obtained from $Sub(p, b-1)$ by taking one more subgraph G_b into account. Initially, $Sub(p, b)$ is set to be $Sub(p, b-1)$. The first step is to choose the best forker instance from $t_{s,1}$, $t_{s,2}$, $\ldots$, $t_{s,p}$ for G_b. Let the best instance be $t_{s,z}$. The second step is to see if $t_{s,z}$ is in $Sub(p, b)$ or not. If not, a condition is checked to decide whether $t_{s,z}$ should be added in or not. Upon the addition of $t_{s,z}$, we may remove some instances and reassign the communications to achieve a lower cost.

We compare two possible results obtained from the above two strategies and assign the one with lower cost to actual $Sub(p, b)$. Hence by computing in a dynamic programming fashion, $Sub(n, B)$ can be obtained. The algorithm and its graphical interpretation are shown in Figure 9.

5.2 Performance Evaluation

The complexity involved in each strategy described in section 5.1 is $O(nB)$. Since the solving of $Sub(n, B)$ needs to invoke $n \times B$ times of strategies 1 and 2, the total complexity of solving

$Sub(n, B)$ by the approximation method is $O(n^2 B^2)$.

We conduct a set of experiments to evaluate the performance of the approximation method. For each problem size (n, B), we randomly generate 50 instances and solve them by using approximation method and exhaustive searching. The data for computation and communication in the experiments are based on the uniform distribution over the range [1,50]. We compare the minimum cost obtained from exhaustive searching (EXHAUST) with those from from approximation (APPROX) and single assignment solution (SINGLE). The optimal single assignment solution is the one in which only one forker instance is allowed. Note that the solutions from SINGLE are obtained from the shortest path algorithm [1]. The results are summarized in Table 4. From the table, we find out that the approximation method yields a tight approximation of the minimum cost. On the contrary, the error range for single copy solution is at least 20%. This again justifies that the replication can lead to a lower cost than an optimal assignment does.

6 Solution of MCRP-SP for computation-intensive applications

6.1 The Solution

Given a computation-intensive application with its SP graph, we generate its parsing tree and assignment graph first. The algorithm finds the minimum weight replication graph from the assignment graph. Then the optimal solution is obtained from the minimum weight replication graph.

The algorithm traverses the parsing tree in the postfix order. Namely, during the traversal, an optimal solution of the subtree S_x, induced by an intermediate node x along with all x's descendant nodes, can be found only after the optimal solutions of x's descendant nodes are found. Given an SP graph G and a distributed system S, we know that there is a one-to-one correspondence between each subtree S_x in a parsing tree $T(G)$ and a limb in the assignment graph of G on S. Whenever a child node b of x is visited, the corresponding limb in the assignment graph will be replaced with a a two-layer T_{chain} limb if b is a T_{chain}- or T_{or}-type node; and a one-layer T_{unit} limb if b is a T_{and}-type node. The algorithm is shown in Table 5. A graphical demonstration of how the algorithm solves the problem is shown in Figure 10.

Before the replacement of a T_{chain} limb is performed (i.e. x is a T_{chain}-type node), each constituent child limb has been replaced with a T_{unit} or two-layer T_{chain} limb. Hence, the shortest path algorithm [1] can be used to compute the weights of the new edges between each node in the source layer and each node in the sink layer of the new T_{chain} limb. The complexity, from lines 05 to 08 of Table 5, in transformation of the limb, corresponding to an intermediate node x with M children, into a two-layer T_{chain} limb is $O(Mn^3)$. An example of illustrating the replacement of a T_{chain} limb is shown from parts (b) to (c) and parts (d) to (e) in Figure 10.

For the replacement of a T_{and} limb, we have to compute $C^b_{p,q}$'s. The values can also be computed by the shortest path algorithm. Hence, the complexity involved in lines 16 and 17 is $O(Bn^3)$. According to the computational model in section 2.2, each task instance s may start its execution if it receives the necessary data from any task instance of its predecessor d. And, from Lemma 2, we know that the minimum sum of initialization costs of multiple task instances of s will be always from only one task instance of d. Therefore, the initialization of task instance $t_{s,p}$ depends on which task instance of d it communicates with. That is why ,in line 19, the communication cost $\mu_{d,s}(i,p)$ is added to the the execution cost of $e_{s,p}$ before $OPT()$ is invoked. And the most significant part of the replacement is to compute the weights on the new edges from the source layer to sink layer. The complexity is $n^2 \times O(OPT())$, which in the worst case is $n^2 2^n$. However, in the average, our OPT function performs pretty well and reduces the complexity significantly. An example of illustrating the replacement of a T_{and} limb is shown from parts (c) to (d) in Figure 10.

We also consider to use the approximation method to find the sub-optimal replacement of a T_{and} limb. In that case, function $OPT()$ in line 21 is replaced with $Sub(n, B)$. The total complexity involved is $O(n^4B^2)$ then.

Finally, for the replacement of a T_{or} limb, if there are B subgraphs connected between the forker and the joiner, then the complexity will be $O(Bn^2)$ for the new edges and $O(Bn^3)$ for $C^b_{p,q}$'s. An example of illustrating the replacement of a T_{or} limb is shown from parts (a) to (b) in Figure 10.

When the traversal reaches the root node of the parsing tree, the result of $FIND()$ will give us either one single layer or two layers, depending on the type of root node. All we have to do is to select the lightest of these n (in single layer) or n^2 (in two layers) shortest path combinations. An optimal replication graph itself is found by combining the shortest paths between the selected nodes that were saved earlier. The whole algorithm has the complexity of

$$O(An^2 2^n) \ + \ \sum_i (R_i n^3) \ + \ \sum_i (C_i n^3)$$

where A is the number of T_{and} limbs, R_i is the number of subgraphs in the ith T_{or} limb, and C_i is the number of layers in the ith T_{chain} limb. This is not greater than $O(Mn^2 2^n)$, where M is the total number of tasks in the SP graph. The complexity of the algorithm is a linear function of M if the number of processors, n, is fixed.

6.2 Conclusion Remark

This paper has focused on MCRP-SP, the optimal replication problem of SP task graphs for computation-intensive applications. The purpose of replication is to reduce inter-processor communication, and to fully utilize the processor power in the distributed systems. The SP graph model, which is extensively used in modeling applications in distributed systems, is used. The applications considered in this paper are computation-intensive in which the execution cost of a task is greater than its communication cost. We prove that MCRP-SP is NP-complete. We present branch-and-bound and approximation methods for SP graphs of type T_{and}. The numerical results show that the algorithm performs very well and avoids a lot of unnecessary searching. Finally, we present an algorithm to solve the MCRP-SP problem for computation-intensive applications. The proposed optimal solution has the complexity of $O(n^2 2^n M)$ in the worst case, while the approximation solution is in the complexity of $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

For the applications in which the communication cost between two tasks is greater than the execution cost of a task, the replication can still be used to reduce the total cost. However, in the extreme case where the execution cost of each task is zero, the optimal allocation will be to assign each task to one processor. We are studying the optimal replication for the general case.

References

[1] S.H. Bokhari, *Assignment Problems in Parallel and Distributed Computing*, Kluwer Academic Publisheds, MA, 1987.

[2] Y. Chen and T. Chen, "Implementing Fault-Tolerance via Modular Redundancy with Comparison," *IEEE Trans. Reliability*, Vol. 39, pp 217-225, June, 1990.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, San Francisco: W.H. Freeman & Company, Publishers, 1979.

[4] R. Jan, D. Liang and S.K. Tripathi, "A Linear-Time Algorithm for Computing Distributed Task Reliability in Pseudo Two-Terminal Series-Parallel Graphs," *submitted for publication to Journal of Parallel and Distributed Computing.*

[5] C.C. Price and S. Krishnaprasad, "Software Allocation Models for Distributed Computing Systems," in *Proc. 4th International Conference on Distributed Computing Systems*, pp 40-48, May 1984.

[6] D. Liang, A.K. Agrawala, D. Mosse, and Y. Shi, "Designing Fault Tolerant Applications in *Maruti*," *Proc. 3rd International Symposium on Software Reliability Engineering,* pp. 264-273, Research Triangle Park, NC, Oct. 1992.

[7] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," in *Proc. 4th International Conference on Distributed Computing Systems*, pp 30-39, May 1984.

[8] P.R. Ma, E.Y.S. Lee and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Trans. Computers*, Vol. C-31, pp 41-47, Jan. 1982.

[9] V.F. Magirou and J.Z. Milis, "An Algorithm for the Multiprocessor Assignment Problem," *Operations Research Letters*, Vol. 8, pp 351-356, Dec. 1989.

[10] C.C. Price and U.W. Pooch, "Search Techniques for a Nonlinear Multiprocessor Scheduling Problem," *Naval Res. Logist. Quart.*, Vol 29, pp 213-233, June 1982.

[11] H.S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithm," *IEEE Trans. Soft. Eng.* Vol 3, pp 85-93, Jan. 1977.

[12] D. Towsley, "Allocating Programs Containing Branches and Loops Within a Multiple Processor System," *IEEE Trans. Software Eng.*, Vol. SE-12, pp. 1018-1024, Oct, 1986.

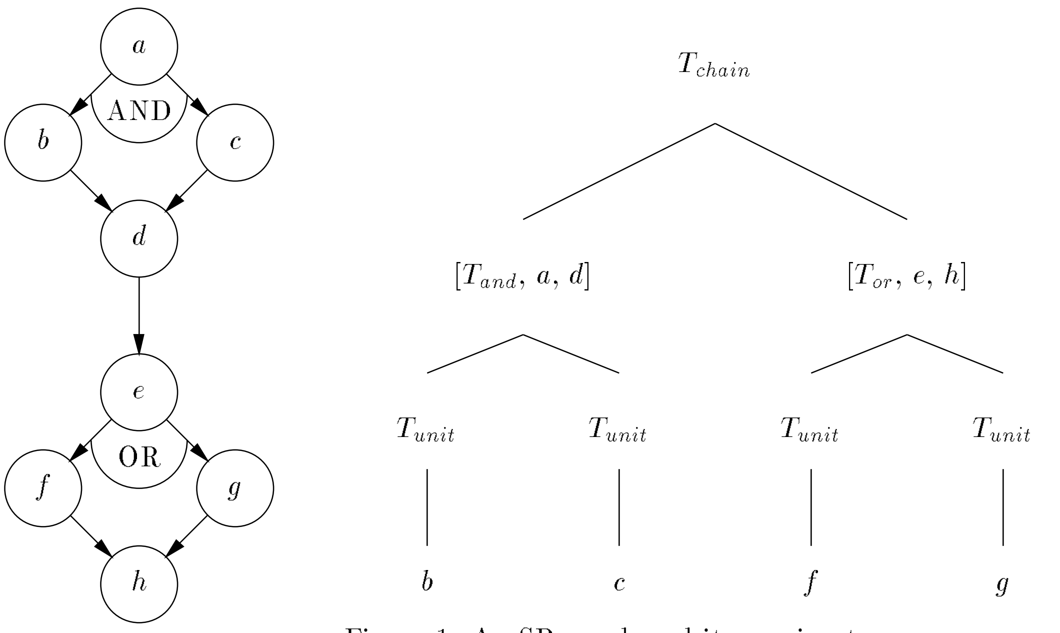
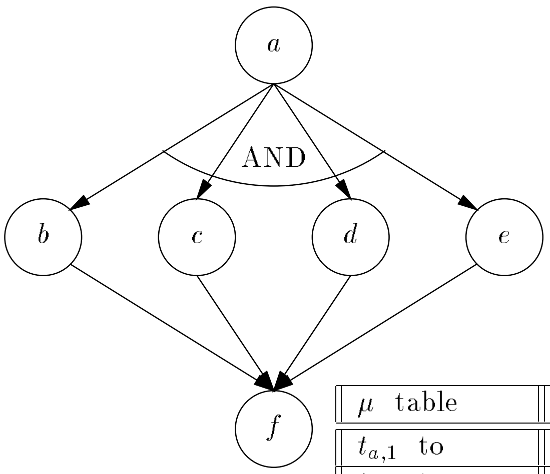


Figure 1: An SP graph and its parsing tree



e table	processor 1	processor 2
task a on	5	5
task b on	7	16
task c on	10	20
task d on	25	8
task e on	14	6
task f on	10	13

μ table	$t_{b,1}$	$t_{b,2}$	$t_{c,1}$	$t_{c,2}$	$t_{d,1}$	$t_{d,2}$	$t_{e,1}$	$t_{e,2}$
$t_{a,1}$ to	1	4	1	4	1	4	1	4
$t_{a,2}$ to	4	1	4	1	4	1	4	1
to $t_{f,1}$ from	3	3	3	3	3	3	3	3
to $t_{f,2}$ from	3	3	3	3	3	3	3	3

Optimal Assignment:

$$e_{a,1} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(1,2) + \mu_{a,e}(1,2) + e_{b,1} + e_{c,1}$$
$$+ e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) + \mu_{e,f}(2,1) + e_{f,1} = 68$$

Optimal Replication:

$$e_{a,1} + e_{a,2} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(2,2) + \mu_{a,e}(2,2) + e_{b,1}$$
$$+ e_{c,1} + e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) + \mu_{e,f}(2,1) + e_{f,1} = 67$$

Figure 2: An example to show how the replication can reduce the total cost

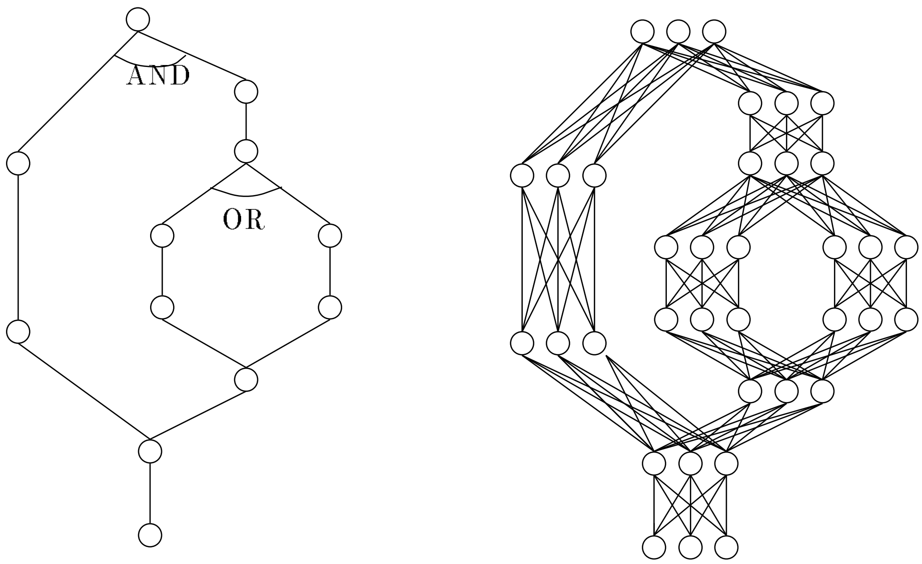


Figure 3: An SP graph and its assignment graph.

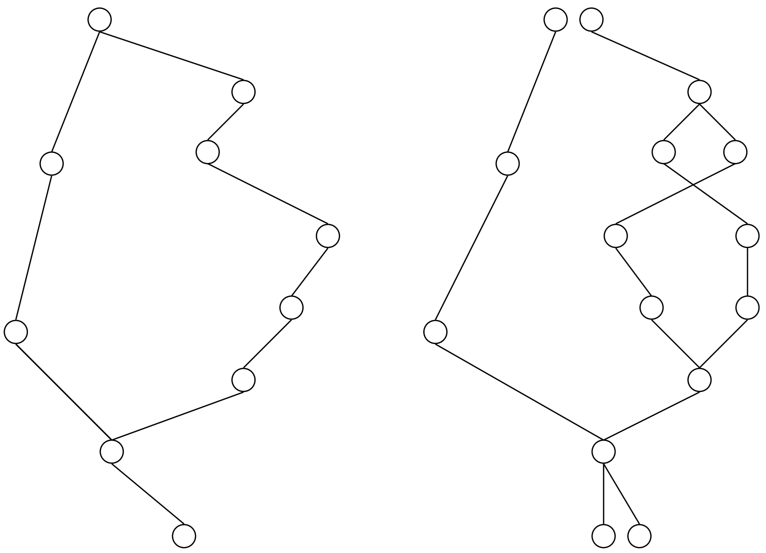


Figure 4: An allocation graph and a replication graph of Figure 3.

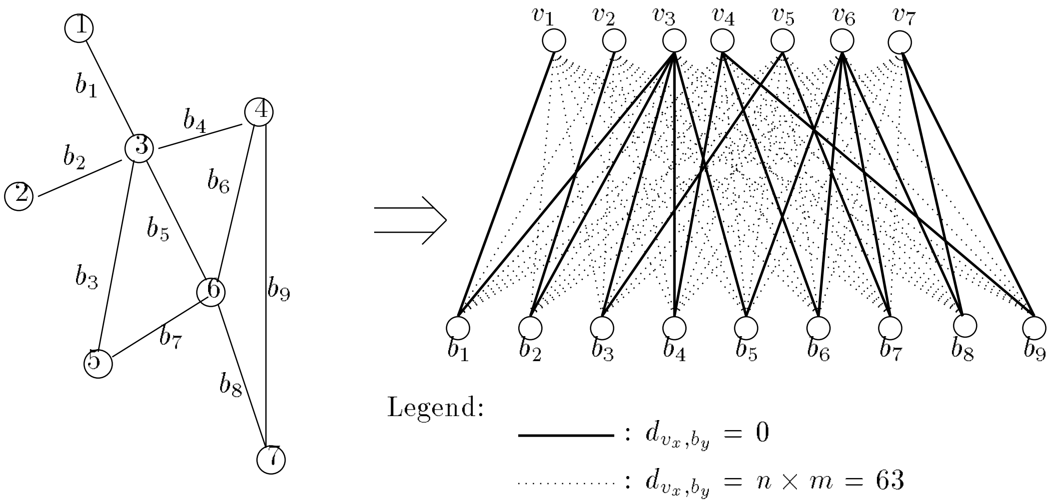


Figure 5: An illustration about how to transform a graph to a UCTR instnace

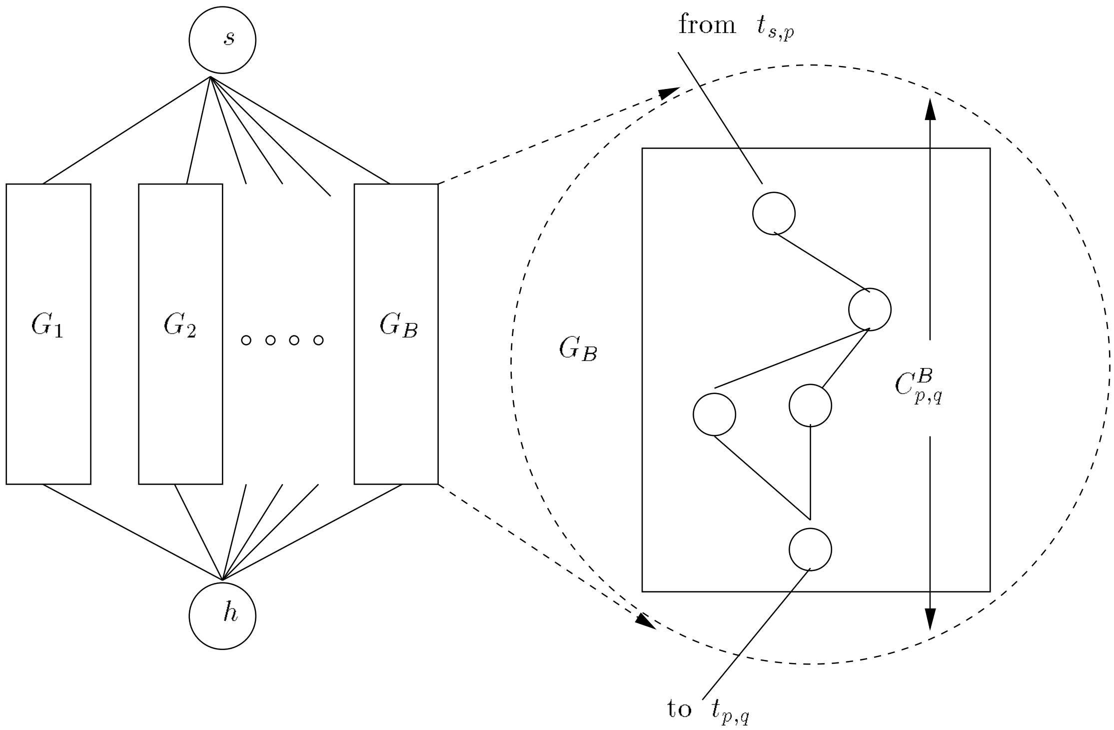
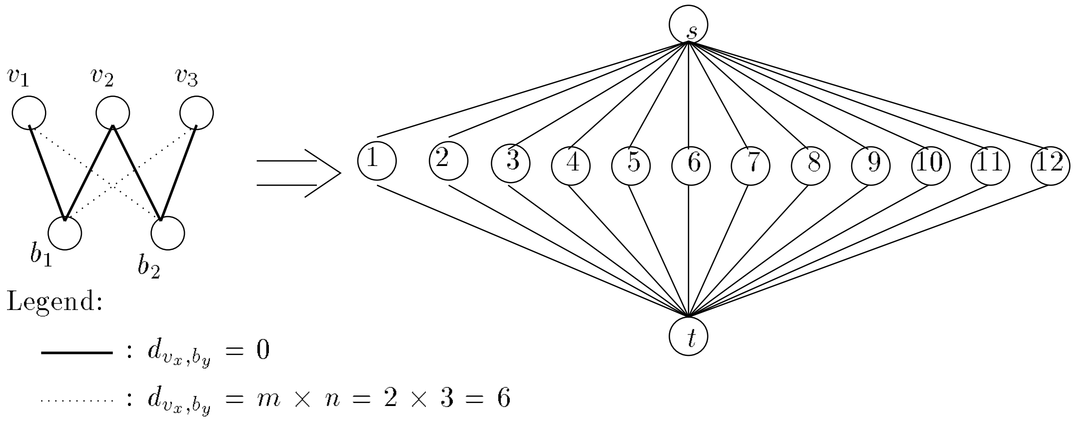


Figure 6: A T_{and} SP graph and the graphical intertrepation of $C^b_{p,q}$.



e table	$p = 1$	$p = 2$	$p = 3$
$e_{s,p} =$	2	2	2
$e_{1,p} =$	0	12	12
$e_{2,p} =$	0	12	12
$e_{3,p} =$	0	12	12
$e_{4,p} =$	0	12	12
$e_{5,p} =$	0	12	12
$e_{6,p} =$	0	12	12
$e_{7,p} =$	0	12	12
$e_{8,p} =$	0	12	12
$e_{9,p} =$	0	12	12
$e_{10,p} =$	0	12	12
$e_{11,p} =$	0	12	12
$e_{12,p} =$	0	12	12
$e_{t,p} =$	0	12	12

μ table	$p = 1$	$p = 2$	$p = 3$
$\mu_{s,1}(p,1) =$	0	0	1
$\mu_{s,2}(p,1) =$	0	0	1
$\mu_{s,3}(p,1) =$	0	0	1
$\mu_{s,4}(p,1) =$	0	0	1
$\mu_{s,5}(p,1) =$	0	0	1
$\mu_{s,6}(p,1) =$	0	0	1
$\mu_{s,7}(p,1) =$	1	0	0
$\mu_{s,8}(p,1) =$	1	0	0
$\mu_{s,9}(p,1) =$	1	0	0
$\mu_{s,10}(p,1) =$	1	0	0
$\mu_{s,11}(p,1) =$	1	0	0
$\mu_{s,12}(p,1) =$	1	0	0
$\mu_{s,i}(p,q) = 0,\ \forall\ 1 \le i \le 12,\ p = 1,2,3$ and $q = 2,3$			
$\mu_{i,t}(p,q) = 0,\ \forall\ 1 \le i \le 12,\ \forall\ 1 \le p, q \le 3$			

Figure 7: An illustration about how to transform a UCTR instance to a T_{and} SP graph

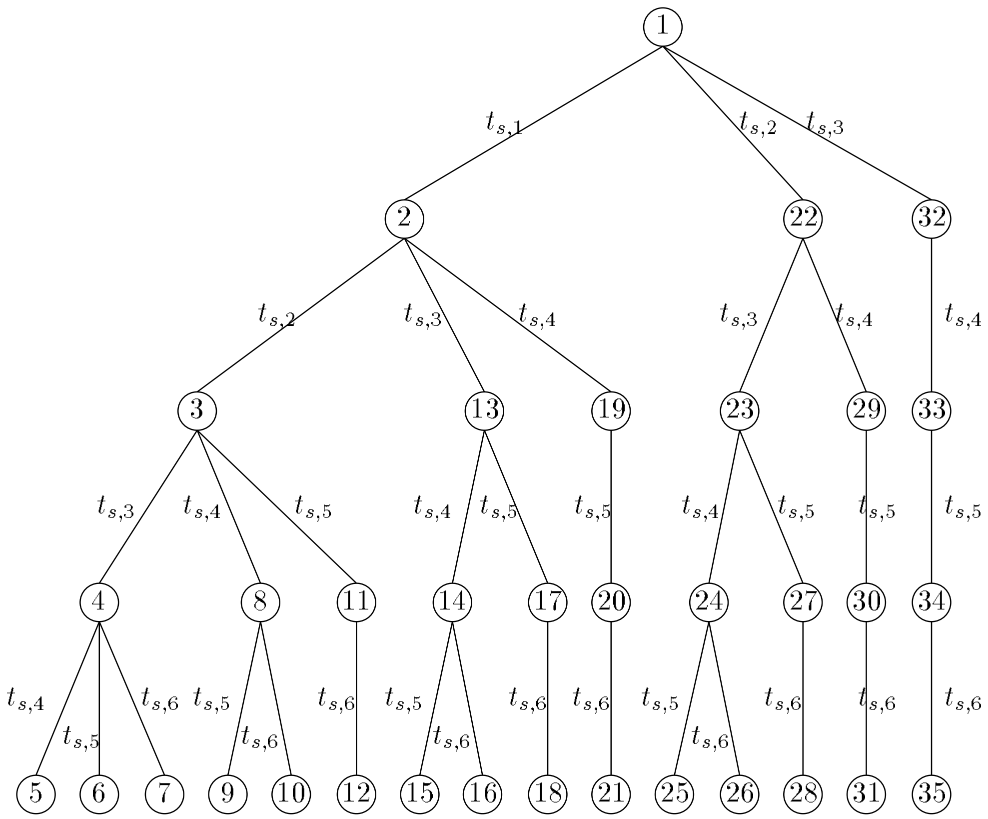


Figure 8: A combination tree for the case where $k = 4$ and $n = 6$

Table 1: **Function** $BB(k, q, \hat{z})$: branch-and-bound algorithm for solving problem $\mathcal{P}_k^q$

Initialize the queue to be empty;
Insert root node v_0 into the queue;
While the queue is not empty **do begin**
    Remove the first node u from the queue;
    Generate all child nodes of u ;
    For each generated child node v **do begin**
        If v is a leaf node (i.e. v is at level k) **then**
            Compute $g(v)$ by setting L to be ϕ ;
            Set $\hat{z} =$ min ($\hat{z}$, $g(v)$);
        else begin /* v is an intermediate node */
            Compute $est(v)$ by (5) ;
            If $est(v) < \hat{z}$ **then**
                Insert v into the queue according to $est(v)$;
        end;
    end;
end;
Return($\hat{z}$).

Table 2: **Function** $OPT(C_{p,q}^b{}'s,\ e_{s,p}'s)$: the optimal solution of MCRP-SP of type T_{and} when $C_{p,q}^b$'s and $e_{s,p}$'s are given

Sort $t_{s,p}$'s into a non-decreasing order by values of $e_{s,p}$'s ;
For $q = 1$ to n **do begin**
    Let node v be a leaf node at level 1;
    Set v to be $t_{s,1}$ and k to be 1;
    Compute $g(v)$ by setting L to be ϕ ;
    Initialize $\hat{z}$ to be $g(v)$;
    For $k = 1$ to n **do**
        $\hat{z} = BB(k, q, \hat{z})$;
    Set $c(q) = \hat{z}$;
end;
Output the combination with the minimum value among $c(1)$, $c(2)$, $\ldots$, $c(n)$.

Figure 9: Pseudo code, graphical demonstration, and dynamic programming table for approximation methods

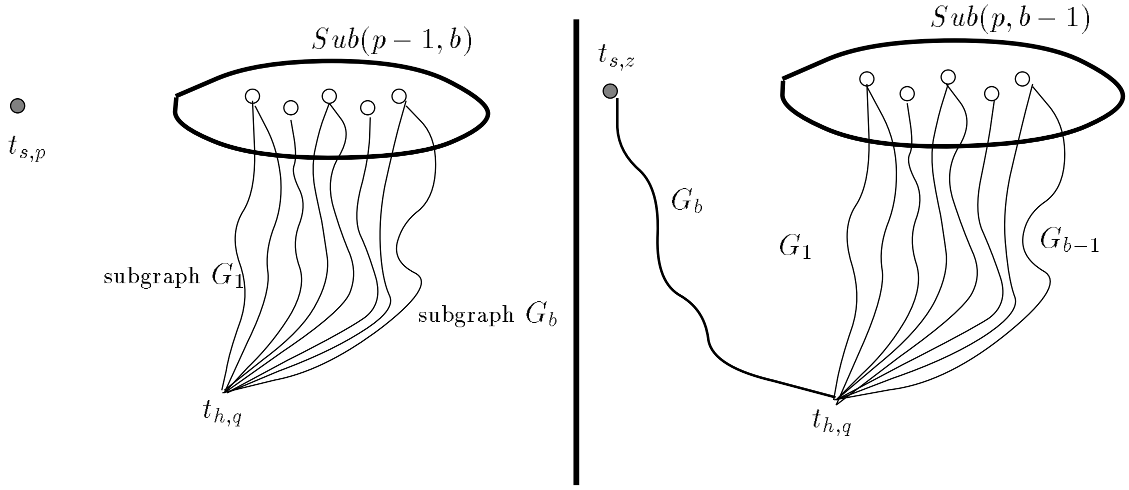
$\underline{Sub(p-1,b) \;\rightarrow\; Sub'(p,b)}$:

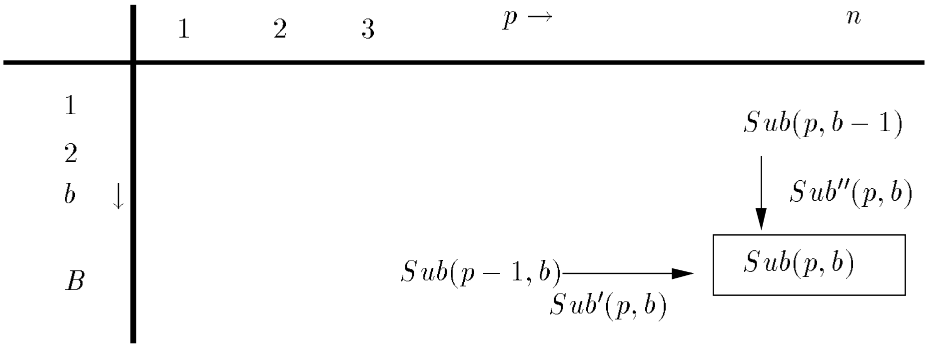
If $e_{s,p} \leq \sum_{i=1}^{b}([\min_{x \in Sub(p-1,b)}(C^{i}_{x,q})] - C^{i}_{p,q})^{+}$
begin
  $Sub'(p,b) = Sub(p-1,b) \oplus t_{s,p}$
  Reassign&Remove($Sub'(p,b)$)
end
Else $Sub'(p,b) = Sub(p-1,b)$

Legend:
  $(x)^{+} = x$, if $x > 0$.
  $(x)^{+} = 0$, if $x \leq 0$.

$\underline{Sub(p,b-1) \;\rightarrow\; Sub''(p,b)}$:

Let $t_{s,z}$ be the one satisfys $\min_{1 \leq i \leq p}(C^{b}_{i,q})$.
If $t_{s,z} \in Sub(p,b-1)$ then
  $Sub''(p,b) = Sub(p,b-1)$
Else
if $e_{s,z} \leq \sum_{i=1}^{b}([\min_{j \in Sub(p,b-1)}(C^{i}_{j,q})] - C^{i}_{z,q})^{+}$
begin
  $Sub''(p,b) = Sub(p-1,b) \oplus t_{s,z}$
  Reassign&Remove($Sub''(p,b)$)
end
Else $Sub''(p,b) = Sub(p,b-1)$





$Sub(p,b) = Min\_Cost(Sub'(p,b), Sub''(p,b))$

Table 3: Computation Results for branch-and-bound approach

n	B	Set I EIM‡	Set I VLF‡	Set II EIM‡	Set II VLF‡	Total Number of leaves (2^n)
4	20	2	6	4	7	16
	24	3	6	3	6	16
	28	4	7	3	6	16
	32	4	7	3	6	16
	36	4	7	4	7	16
	40	3	6	3	6	16
8	20	36	74	16	51	256
	24	40	75	21	62	256
	28	50	86	26	68	256
	32	63	94	37	78	256
	36	73	96	47	84	256
	40	81	97	50	86	256
12	20	186	558	81	340	4,096
	24	231	639	102	398	4,096
	28	349	839	167	543	4,096
	32	451	967	204	617	4,096
	36	454	984	269	720	4,096
	40	636	1,186	301	780	4,096
16	20	758	3,216	203	1,175	65,536
	24	1,065	4,161	329	1,711	65,536
	28	1,335	4,862	546	2,496	65,536
	32	1,884	6,250	726	3,127	65,536
	36	2,322	7,227	839	3,493	65,536
	40	2,880	8,511	1,179	4,510	65,536
20	20	2,026	12,042	389	3,079	1,048,576
	24	3,579	18,866	761	5,280	1,048,576
	28	5,551	27,018	1,227	7,905	1,048,576
	32	6,405	30,521	1,709	10,357	1,048,576
	36	9,517	40,767	2,681	15,032	1,048,576
	40	11,651	48,087	3,086	16,857	1,048,576

‡: Each value shown is the average value over 50 runs.

Table 4: Simulation Results for Approximation Method

n	B	SINGLE‡	APPROX‡	EXHAUST‡	single error %	approx error %
4	20	2876	2407	2400	20	0.28
	24	3463	2835	2831	22	0.16
	28	4032	3264	3259	24	0.18
	32	4606	3678	3673	25	0.11
	36	5198	4084	4082	27	0.05
	40	5790	4514	4514	28	0.00
8	20	2794	2282	2250	24	1.46
	24	3356	2672	2636	27	1.38
	28	3931	3060	3028	30	1.05
	32	4540	3443	3413	33	0.88
	36	5127	3831	3800	35	0.80
	40	5683	4215	4192	36	0.55
12	20	2767	2213	2161	28	2.42
	24	3359	2592	2542	32	1.99
	28	3912	2996	2941	33	1.88
	32	4491	3364	3299	36	1.97
	36	5063	3736	3676	38	1.62
	40	5610	4101	4043	39	1.43
16	20	2733	2167	2111	29	2.66
	24	3287	2558	2492	32	2.66
	28	3844	2932	2865	34	2.31
	32	4393	3315	3240	36	2.32
	36	4991	3659	3584	39	2.10
	40	5558	4045	3970	40	1.89

$\ddagger$: Each value shown is the average value over 50 runs.

$$\text{single error\%} = \frac{\text{SINGLE} - \text{EXHAUST}}{\text{EXHAUST}} \times 100\%.$$

$$\text{approx error\%} = \frac{\text{APPROX} - \text{EXHAUST}}{\text{EXHAUST}} \times 100\%.$$

Table 5: **Algorithm** $FIND(S_x)$: the algorithm for finding the shortest path combinations from the limb which corresponds to the subtree S_x induced by an intermediate node x and all x's descendant nodes in a parsing tree

01 **Case** of the type of intermediate node x:
02 Type T_{chain} :
03 **For** b = the $first$ child node of x **to** the $last$ one **do**
04 $FIND(S_b)$; /* Now the limb corresponding to S_b is replaced */
05 Replace the limb corresponding to S_x with a two-layer T_{chain} limb where
06 the source (sink) layer of the old limb is the source (sink) layer of new 2-layer limb;
07 Put weights on the edges between source and sink layers equal to the shortest path
08 between the corresponding nodes;
09
10 Type T_{and} : /* Let x = [T_{and}, forker s, joiner h] */
11 Let d be the predecessor of forker s in G (i.e. $< d, s > \in V$);
12 Let B be the number of child nodes of x in the parsing tree;
13 /* I.e. there are B subgraphs connected by s and h */
14 **For** b = the first child node of x **to** the B-th child of x **do**
15 $FIND(S_b)$; /* Now the limb corresponding to S_b is replaced */
16 **For** $p = 1$ **to** n, $q = 1$ **to** n **and** $b = 1$ **to** B **do**
17 Compute the minimum replication cost $C^b_{p,q}$ from $t_{s,p}$ to $t_{h,q}$ w.r.t. child b ;
18 **For** $i = 1$ **to** n **do begin**
19 **For** $p = 1$ **to** n **do** $E_{s,p} = \mu_{d,s}(i,p) + e_{s,p}$;
20 /* $E_{s,p}$ accounts for initialization by $t_{d,i}$ and execution cost itself. */
21 **For** $q = 1$ **to** n **do** $\mu_{d,h}(i,q) = OPT({C^b_{p,q}}'s, {E_{s,p}}'s)$;
22 /* Create new edges from $t_{d,i}$'s to $t_{h,q}$'s */
23 **end**;
24 Replace the T_{and} limb with a T_{unit} limb, where source layer = sink layer = layer h,
25 and there are new edges from layer d to layer h;
26
27 Type T_{or} : /* Let x = [T_{or}, forker s, joiner h] */
28 Use the same method described above from lines 12 to 17 to compute $C^b_{p,q}$'s ;
29 Replace the T_{or} limb with a two-layer T_{chain} limb, where
30 the source (sink) layer of T_{or} limb is the source (sink) layer of T_{chain} limb and
31 $\mu_{s,h}(p,q) = \min_b(C^b_{p,q})$, $\forall$ p and q ;
32 **end case**;
33 Save the shortest paths between any node in source layer and any node
in sink layer for future reference.

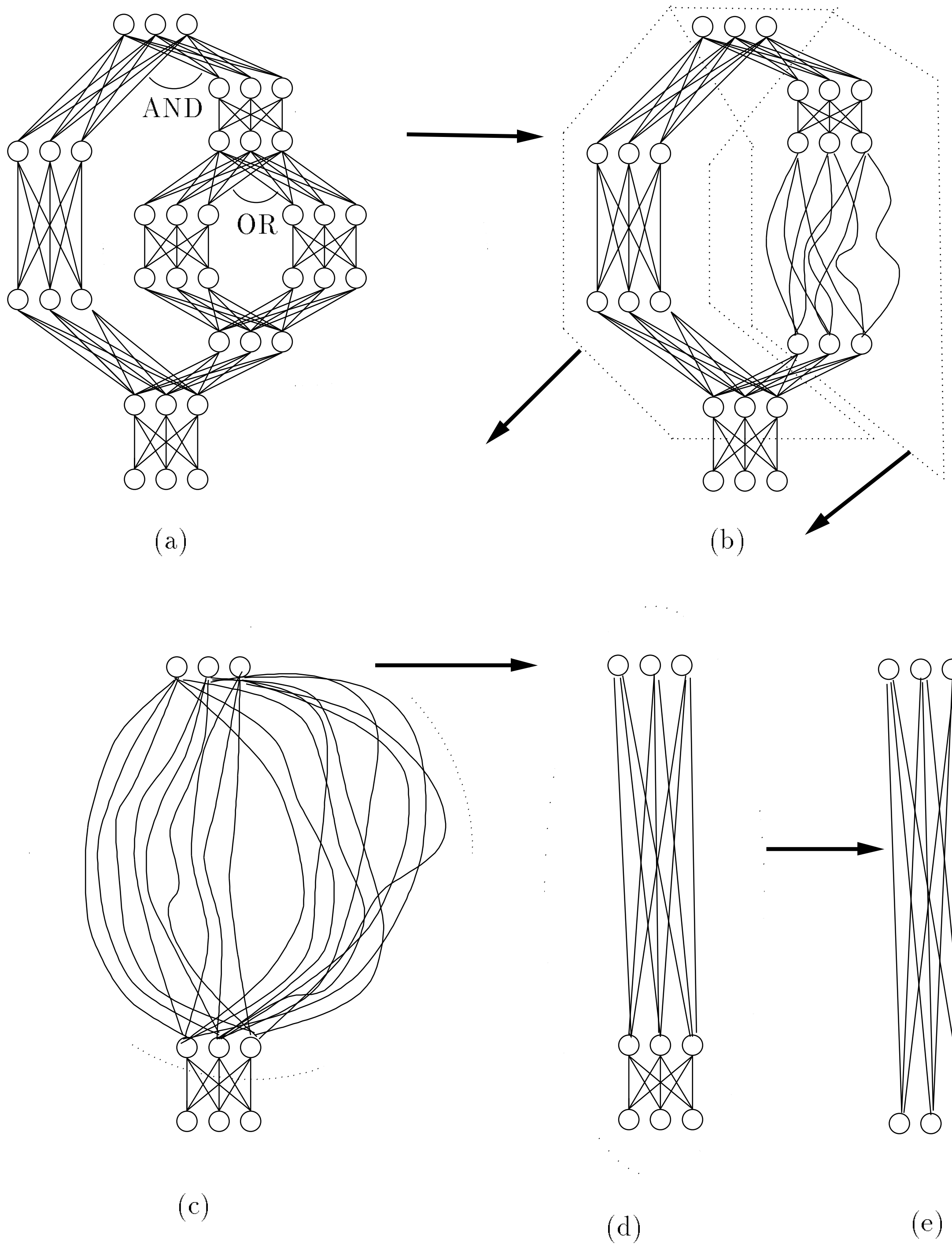


Figure 10. A graphical demostration of how to find an optimal solution for MCRP-SP