Information Processing Letters 32 (1989) 61–71
North-Holland

24 July 1989

AN LC BRANCH-AND-BOUND ALGORITHM FOR THE MODULE ASSIGNMENT PROBLEM

Maw-Sheng CHERN

Department of Industrial Engineering, National Tsing Hua University, Hsinchu, Taiwan, Rep. China

G.H. CHEN and Pangfeng LIU

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Rep. China

Communicated by K. Ikeda
Received 20 July 1988
Revised 10 March 1989

Distributed processing has been a subject of recent interest due to the availability of computer networks. Over the past few years it has led to the identification of several challenging problems. One of these is the problem of optimally distributing program modules over a distributed processing system. In this paper we present an LC (Leas Cost) branch-and-bound algorithm to find an optimal assignment that minimizes the sum of execution costs and communication costs. Experimental results show that, for over half of the randomly generated instances, the saving rates exceed 99%. Moreover, it appears that the saving rates rise as the size of the instances increases. Finally, we also introduce two reduction rules to improve the efficiency of the algorithm for some special cases.

Keywords: Branch-and-bound algorithms, module assignment problem, distributed processing system

1. Introduction

Distributed processing has been a subject of recent interest due to the availability of computer networks. Over past few years it has led to the identification of several challenging problems. One of these is the module assignment problem. Briefly, the problem can be stated as follows. Given a set of m program modules to be executed on a distributed system of n processors, to which processor should each module be assigned? The problem for more than three processors is known to be NP-hard [4]. The program modules may be viewed as program segments or subroutines, and control is transferred between program modules through subroutine calls. Two costs are considered in the problem: execution cost and communication cost. The execution cost is the cost of executing program modules; the communication cost is the cost of communication among processors. The communication cost is actually caused due to the necessary data transmission among program modules. If a program module is not executable on a particular processor, the corresponding running cost is taken to be infinity.

The module assignment problem has been studied extensively for various models, for example see [3,5–7,9–11,13,14,16], and polynomial-time solutions have been obtained for some restricted cases, such as two processors [1,12,15], tree structure of interconnection pattern of program modules [1,4], and fixed communication cost [2]. In this paper, we consider a general model where the number of processors may be any value, the interconnection pattern of program modules may be any structure, and, more important, various constraints such as storage constraint and load constraint may be included. We then present an LC

(Least Cost) branch-and-bound algorithm to find an optimal assignment that minimizes the sum of execution costs and communication costs. The effectiveness of the algorithm is shown by experimental results. Moreover, we introduce two reduction rules to improve the efficiency of the algorithm for some special cases.

2. The model

The model we consider for the module assignment problem is as follows.

(i) There are m program modules $M_1, M_2, \ldots, M_m$ to be executed.
(ii) There are n processors $P_1, P_2, \ldots, P_n$ available.
(iii) $E(M_i, P_j)$ is the cost of executing program module M_i on processor P_j.
(iv) $T(M_i, M_j, P_r, P_s)$ is the communication cost that is incurred by program modules M_i and M_j when they are assigned to processor P_r and P_s respectively. If M_i and M_j are assigned to the same processor, the communication cost between them is assumed to be 0.
(v) The objective is to minimize the sum of the execution costs and communication costs.
(vi) There are the following constraints:

(1) *Storage constraint:* The available storage provided by a processor is limited. Let $SUB(P_r)$ denote the storage limit for processor P_r and $STOR(M_{r_i})$ denote the amount of storage occupied by program module M_{r_i}. If $M_{r_1}, M_{r_2}, \ldots, M_{r_k}$ are all assigned to P_r, then $STOR(M_{r_1}) + STOR(M_{r_2}) + \cdots + STOR(M_{r_k})$ must not exceed $SUB(P_r)$.
(2) *Load constraint*: The available computational load provided by a processor has an upper bound. Let $LUB(P_r)$ denote the computational load upper bound for processor P_r and $LOAD(M_{r_i})$ denote the computational load required by program module M_{r_i}. If $M_{r_1}, M_{r_2}, \ldots, M_{r_k}$ are all assigned to P_r, then $LOAD(M_{r_1}) + LOAD(M_{r_2}) + \cdots + LOAD(M_{r_k})$ must not exceed $LUB(P_r)$.
(3) Some subsets of program modules are restricted to the same processor.
(4) Some program modules are restricted to some specific processors.

More constraints can be included in the model if necessary. Based on the model, the problem can be mathematically formulated as follows. Let M denote the set of program modules, P the set of processors, and let ψ be a mapping from M to P, i.e., $\psi: M \to P$. The problem is to minimize

$$\sum_{i=1}^{m} E(M_i, \psi(M_i)) + \sum_{i=1}^{m} \sum_{j=i+1}^{m} T(M_i, M_j, \psi(M_i), \psi(M_j)) \quad \text{over all feasible mappings } \psi.$$

3. A branch-and-bound algorithm

Since a tree structure is a convenient representation of the execution of the branch-and-bound algorithm, we will describe the branch-and-bound algorithm through the generation of the branch-and-bound tree. Each edge in the branch-and-bound tree represents an assignment of a program module to some processor. The nodes at the lowest (mth) level represent complete solutions and all the other nodes represent partial solutions. A node is infeasible if it does not satisfy the constraints, and an assignment of a program module to some processor is infeasible if it leads to an infeasible node. It is clear that if a node is infeasible, then all of its descendants are infeasible. This suggests that whenever an infeasible node is detected, it may be fathomed, thereby preventing the generation of its subtree. For each node, a pair of values U_{ij} and D_{ij} are estimated for all free (not yet assigned) program modules M_i and all processors P_j. U_{ij} is the minimum increasing cost that is expected for all the free program modules, given that M_i is to be

assigned to P_j. Let S denote the set of program modules that are included in the corresponding partial solution. U_{ij} is estimated as follows:

$$U_{ij} = E(M_i, P_j) + \sum_{M_x \in S} T(M_i, M_x, P_j, P_{\phi(x)}) + \sum_{M_y \notin S + \{M_i\}} \min\Big\{ E(M_y, P_r) + \sum_{M_z \in S} T(M_y, M_z, P_r, P_{\phi(z)}) + T(M_y, M_i, P_r, P_j) \mid r \in \{1, 2, \ldots, n\} \text{ and } M_y \to P_r \text{ is feasible} \Big\},$$

where $P_{\phi(x)}$ and $P_{\phi(z)}$ are the respective processors to which M_x and M_z were assigned. U_{ij} is set to infinity if $M_i \to P_j$ is infeasible or $M_y \to P_r$ is infeasible for all processors P_r. On the other hand, D_{ij} is the minimum increasing cost that is expected for all the free program modules, given that M_i is not to be assigned to P_j. D_{ij} is estimated as follows:

$$D_{ij} = \min\{ U_{ik} \mid k \in \{1, 2, \ldots, n\},\ k \neq j \}.$$

In addition to U_{ij} and D_{ij}, two costs, current cost (CC) and expected cost (EC), are computed for each node. The current cost is the cost of the partial solution (complete solution, if the node is at the lowest level) that is represented by the node. Initially, $CC = 0$ for the root node. For a nonroot node, if the edge to it from its parent node represents the assignment of M_i to P_j, the current cost is computed by the following equation:

$$CC = CC_p + E(M_i, P_j) + \sum_{M_k \in S} T(M_i, M_k, P_j, P_{\phi(k)}),$$

where CC_p denotes the current cost of the parent node, S denotes the set of program modules that are included in the partial solution represented by the parent node, and $P_{\phi(k)}$ is the processor to which M_k was assigned. Note that for each feasible node at the lowest level, CC is the cost of the corresponding complete solution. On the other hand, the expected cost is the minimum increasing cost that is expected for all the free program modules. It is computed as follows:

$$EC = \min\{ U_{ij} \mid \text{all free program modules } M_i \text{ and all processors } P_j \}.$$

Thus, for each node, $CC + EC$ is a lower bound on the costs of the complete solutions that will appear in its subtree.

The generation of the branch-and-bound tree starts at the root and follows the LC (Least Cost) strategy [8]. The nodes that wait to be branched are called live nodes. The live node that has a minimum value of $CC + CE$ is selected to be branched next. If a tie exists, break any. When a node is branched, the program module M_i that satisfies $EC = U_{ij}$ for some processor P_j will be assigned to all processors (see Fig. 1). An upper bound cost (UC) is along with the branch-and-bound algorithm and it represents an upper bound on the optimal cost. UC is set to be infinity intially and is updated to be $\min\{UC, CC\}$ whenever a feasible node at the lowest level is reached. If a node satisfies $CC + EC \geqslant UC$, then it is fathomed, since further branching from it will not lead to better solutions. If a node satisfies $CC + U_{ij} \geqslant UC$ for some free program module M_i and some processor P_j, then it is impossible to get better solutions if M_i is to be assigned to P_j. This implies that M_i should not be assigned to P_j (denoted by $M_i \nrightarrow P_j$). Thus a node can be fathomed if it satisfies $CC + U_{ij} \geqslant UC$ for some free program module M_i and all processors P_j. Similarly, if a node satisfies $CC + D_{ij} \geqslant UC$ for some free program module M_i and some processor P_j, then M_i is restricted to P_j (denoted by $M_i \to P_j$). These restrictions will cause more nodes to be fathomed.

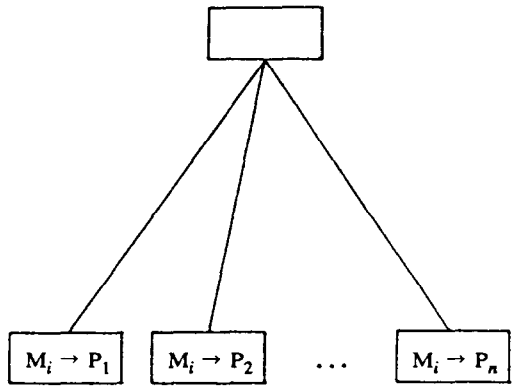

Fig. 1. The branching of a node.

Finally, when the branch-and-bound algorithm terminates, the value of UC is the optimal cost and the optimal solutions will be generated at the nodes with $CC = UC$.

The branch-and-bound algorithm is as follows.

Branch-and-bound algorithm

(1) set live-node-list to be empty; {live-node-list is a priority queue storing live nodes}
(2) put root node into live-node-list and compute U_{ij}, D_{ij}, EC, and CC for root node;
(3) set UC to be infinity;
(4) **while** live-node-list is not empty **do**
begin
(5) choose a node α with minimum value of $CC + EC$ from live-node-list;
(6) **if** α has $CC + EC \geqslant UC$ or $CC + U_{ij} \geqslant UC$ for some free M_i and all P_j, $j = 1, 2, \ldots, n$
(7) **then** remove α from live-node-list
else begin
(8) (9) **if** $CC + U_{ij} \geqslant UC$ for some free M_i and some P_j **then** a restriction "$M_i \nrightarrow P_j$" is added;
(10) (11) **if** $CC + D_{ij} \geqslant UC$ for some free M_i and some P_j **then** a restriction "$M_i \rightarrow P_j$" is added;
(12) branch α (assign M_i that satisfies $EC = U_{ij}$ for some P_j to all processors);
(13) **for** each (assume β) of newly generated nodes **do**
(14) **if** β is feasible **then begin**
(15) compute U_{ij}, D_{ij}, EC, and CC;
(16) **if** β is at the lowest level
(17) (18) **then if** $CC < UC$ **then** replace UC with CC
(19) (20) **else if** $CC + EC < UC$ **then** insert β into live-node-list
end;
(21) remove α from live-node-list
end
end;
(22) output UC and the nodes (at the lowest level) with $CC = UC$.

4. Experimental results

In this section we provide experimental results to show the effectiveness of the branch-and-bound algorithm. The criterion we adopt to evaluate the performance of the algorithm is the saving rate, which is

Table 1
Saving rates for randomly generated instances

		COMMUNICATION COSTS								
		[0,10]			[0,50]			[0,100]		
n	m	p=0.2	p=0.5	p=0.8	p=0.2	p=0.5	p=0.8	p=0.2	p=0.5	p=0.8
2	8	0.96047	0.94324	0.94011	0.95107	0.93072	0.91115	0.92915	0.92681	0.95655
	9	0.97087	0.95992	0.94584	0.93919	0.91964	0.94701	0.93137	0.94819	0.97204
	10	0.98114	0.96668	0.92447	0.92701	0.95789	0.97743	0.95280	0.96844	0.98114
	11	0.98539	0.97152	0.96683	0.95228	0.97015	0.98432	0.94661	0.97475	0.99291
	12	0.99313	0.98723	0.98151	0.98244	0.97384	0.98986	0.96574	0.98234	0.99499
	13	0.98985	0.98431	0.96492	0.97630	0.98292	0.99671	0.98057	0.99298	0.99737
	14	0.99552	0.98761	0.96805	0.97910	0.98860	0.99511	0.98453	0.99705	0.99822
	15	0.99419	0.98428	0.98315	0.98531	0.99272	0.99681	0.98669	0.99860	0.99894
	16	0.99732	0.98750	0.97985	0.98997	0.99576	0.99841	0.98377	0.99871	0.99956
	17	0.99773	0.98429	0.98527	0.99602	0.99685	0.99922	0.99282	0.99947	0.99968
	18	0.99713	0.99446	0.99496	0.98682	0.99847	0.99935	0.98794	0.99950	0.99980
	19	0.99862	0.99207	0.98998	0.99391	0.99868	0.99983	0.99741	0.99961	0.99992
	20	0.99888	0.99738	0.99420	0.99533	0.99934	0.99986	0.99819	0.99981	0.99995
3	5	0.95109	0.95109	0.94285	0.95109	0.92967	0.93461	0.91483	0.92802	0.93626
	6	0.97767	0.96669	0.96614	0.97054	0.94473	0.96120	0.94309	0.95846	0.96669
	7	0.98615	0.98414	0.97298	0.97774	0.95634	0.96274	0.98634	0.96969	0.98908
	8	0.99489	0.99325	0.99325	0.98831	0.97959	0.98611	0.98343	0.98252	0.99410
	9	0.99830	0.99634	0.99271	0.99242	0.98586	0.99592	0.98442	0.99100	0.99734
	10	0.99867	0.99789	0.99525	0.98919	0.98936	0.99701	0.99087	0.99728	0.99915
	11	0.99956	0.99802	0.99744	0.99502	0.99584	0.99933	0.99384	0.99787	0.99946
	12	0.99970	0.99765	0.99676	0.99667	0.99687	0.99961	0.99334	0.99900	0.99987
	13	0.99955	0.99896	0.99710	0.99782	0.99796	0.99990	0.99918	0.99950	0.99992
4	4	0.94780	0.94780	0.94545	0.94310	0.90557	0.93372	0.94310	0.93137	0.94545
	5	0.98344	0.97289	0.97465	0.97992	0.97875	0.97172	0.97113	0.96058	0.95882
	6	0.99454	0.99176	0.99088	0.98736	0.98443	0.98472	0.99102	0.97462	0.98209
	7	0.99775	0.99574	0.99515	0.99453	0.99003	0.99090	0.99398	0.99043	0.99306
	8	0.99923	0.99789	0.99750	0.99347	0.99165	0.99828	0.99217	0.99622	0.99752
	9	0.99957	0.99904	0.99871	0.99827	0.99671	0.99753	0.99495	0.99564	0.99951
	10	0.99982	0.99957	0.99883	0.99852	0.99736	0.99933	0.99869	0.99912	0.99971
5	3	0.89743	0.89743	0.89102	0.88461	0.87820	0.87820	0.89743	0.89743	0.89743
	4	0.97055	0.96414	0.96542	0.96927	0.96670	0.95390	0.96414	0.96158	0.95006
	5	0.99078	0.99052	0.98822	0.99180	0.98259	0.97388	0.98899	0.97772	0.98745
	6	0.99790	0.99703	0.99278	0.99170	0.98709	0.98540	0.99441	0.99139	0.99267
	7	0.99894	0.99858	0.99811	0.99751	0.99183	0.99298	0.99543	0.99145	0.99798
	8	0.99981	0.99955	0.99862	0.99832	0.99833	0.99725	0.99943	0.99675	0.99879
	9	0.99993	0.99971	0.99889	0.99901	0.99835	0.99924	0.99849	0.99898	0.99979
6	3	0.92664	0.92664	0.92664	0.92200	0.92664	0.92200	0.92200	0.92664	0.92200
	4	0.98083	0.98392	0.97852	0.98315	0.97543	0.96540	0.97852	0.97389	0.96308
	5	0.99629	0.99372	0.99539	0.99217	0.98883	0.98626	0.99320	0.98844	0.99191
	6	0.99921	0.99820	0.99794	0.99440	0.99460	0.99631	0.99625	0.99610	0.99687
	7	0.99937	0.99864	0.99886	0.99763	0.99801	0.99620	0.99867	0.99860	0.99838
	8	0.99992	0.99961	0.99885	0.99962	0.99788	0.99926	0.99960	0.99811	0.99975

defined to be $1 - (N(n-1)/(n^{m+1}-1))$, where N is the number of nodes that are actually generated by the branch-and-bound algorithm and $(n^{m+1}-1)/(n-1)$ is the number of nodes in the complete (unbounded) tree. Since the constraints introduced in the model have a great influence on the saving rate, we do not take them into account in the experiment (it is clear that we would get higher saving rates if the constraints were considered). We make the following assumptions in the experiment:

(1) Execution costs are given randomly from [0,100].
(2) Communication costs are given randomly from [0,10], [0,50], and [0,100].
(3) Any two program modules have the same probability p to communicate with each other during program execution. $p = 0.2$, 0.5, 0.8 are considered.
(4) Five instances are run for each case and the average saving rate is computed.

Table 1 shows the resulting saving rates. For over half of the generated instances the saving rates exceed 99%, and it appears that the saving rates will be higher for larger-size instances.

Besides, we also compare optimal costs with those obtained by the random assignment method. The random assignment method randomly assigns each program module to a processor. We assume that execution costs and communication costs are given randomly from [0,100] and [0,50] respectively. There is a load constraint on each processor. The $LOAD(M_i)$'s are given randomly from [0,50] and each processor has the same value of

$$LUB = \frac{3}{5} * \sum_{i=1}^{m} LOAD(M_i).$$

Table 2
Comparison of optimal costs with costs obtained by random assignment method

	Optimal cost	Random assignment method			
		Number of successes	Greatest cost	Least cost	Average cost
	172	38	482	185	370
	161	34	574	192	366
$m = 4$	115	31	415	131	212
	108	21	370	119	258
	89	23	425	188	298
	183	37	628	196	397
	147	37	476	167	330
$m = 5$	119	23	722	144	471
	173	32	560	217	411
	172	38	621	259	440
	193	36	678	335	471
	202	25	547	255	419
$m = 6$	130	20	589	319	452
	147	32	646	283	495
	45	31	581	282	410
	208	35	871	351	618
	164	34	803	298	558
$m = 7$	149	29	733	290	491
	110	24	497	225	389
	215	34	813	375	564
	260	35	993	531	798
	175	22	899	397	544
$m = 8$	207	36	985	460	687
	150	32	885	370	655
	287	31	1255	665	937
	260	38	1158	561	867
	272	19	1042	629	849
$m = 9$	222	28	1105	506	749
	253	39	1126	451	851
	106	31	1010	388	688
	296	35	1255	691	951
	73	24	1289	458	924
$m = 10$	220	32	1250	642	977
	339	31	1478	869	1103
	375	33	1619	927	1266

The experiment is performed for $n = 4$, $m = 4, 5, \ldots, 10$, and $p = 0.5$. Five instances are randomly generated for each value of m, and 50 random assignments are made for each instance. Among the 50 random assignments, the number of successful (feasible) assignments is computed, and the greatest cost, the least cost, and the average cost are given for the successful assignments. Table 2 shows the experimental results.

5. Two reduction rules

If constraints do not cause interference among program modules (for example, the first three constraints in the model cause interference among program modules), the following two reduction rules can be implemented for fathoming more nodes. Let C_{ij} denote the maximum communication cost that is incurred by M_i, given that M_i is to be assigned to P_j. That is,

$$C_{ij} = \sum_{t=1, t\neq i}^{m} \max\{T(M_i, M_t, P_j, P_r) \mid r = 1, 2, \ldots, n\}.$$

We then have the following property.

Property 1. *If* $E(M_i, P_k) \geq E(M_i, P_j) + C_{ij}$ *for* $k = 1, 2, \ldots, j-1, j+1, \ldots, n$, *then there exists an optimal assignment in which* M_i *is assigned to* P_j.

Proof. Suppose that ψ^* is an optimal assignment in which M_i is assigned to P_v $(v \neq j)$. Then we can construct an assignment ψ' from ψ^* by changing the assignment of M_i from P_v to P_j. Let Δ^* be the total cost of ψ^* and Δ' be the total cost of ψ'. We have

$$\begin{aligned}
\Delta' - \Delta^* &= E(M_i, P_j) + \sum_{r=1, r\neq i}^{m} T(M_i, M_r, P_j, \psi'(M_r)) \\
&\quad - E(M_i, P_v) - \sum_{r=1, r\neq i}^{m} T(M_i, M_r, P_v, \psi^*(M_r)) \\
&\leq E(M_i, P_j) - E(M_i, P_v) + \sum_{r=1, r\neq i}^{m} T(M_i, M_r, P_j, \psi'(M_r)) \\
&\leq E(M_i, P_j) - E(M_i, P_v) + C_{ij} \leq 0.
\end{aligned}$$

So, ψ' is an optimal assignment. □

Suppose that M_i only communicates with M_u. Let $SS_{\max}$ denote the maximum total cost that is incurred by the set $\{M_i, M_u\}$, given that M_i and M_u are to be assigned to the same processor, and $SR_{\min}$ denote the minimum total cost that is incurred by the set $\{M_i, M_u\}$, given that M_i and M_u are to be assigned to different processors. That is,

$$SS_{\max} = \max\{E(M_i, P_j) + E(M_u, P_j) \mid j = 1, 2, \ldots, n\},$$

$$SR_{\min} = \min\{E(M_i, P_j) + E(M_u, P_v) + T(M_i, M_u, P_j, P_v) \mid j, v \in \{1, 2, \ldots, n\} \text{ and } j \neq v\}.$$

We then have the following property.

Property 2. *If M_i only communicates with M_u and $SS_{max} \leqslant SR_{min}$, then there exists an optimal assignment in which both M_i and M_u are assigned to the same processor.*

Proof. Suppose that ψ^* is an optimal assignment in which M_i and M_u are assigned to P_j and P_v, $j \neq v$, respectively. Then we can construct an assignment ψ' from ψ^* by changing the assignment of M_i from P_j to P_v. Let Δ^* be the total cost of ψ^* and Δ' be the total cost of ψ'. We have

$$\begin{aligned}\Delta' - \Delta^* &= E(M_i, P_v) - \left[E(M_i, P_j) + T(M_i, M_u, P_j, P_v)\right] \\ &= \left[E(M_i, P_v) + E(M_u, P_v)\right] - \left[E(M_i, P_j) + E(M_u, P_v) + T(M_i, M_u, P_j, P_v)\right] \\ &\leqslant SS_{max} - SR_{min} \leqslant 0\end{aligned}$$

So, ψ' is an optimal assignment. □

6. Concluding remarks

Sinclair [14] has proposed a branch-and-bound algorithm for solving the module assignment problem with the same model except that no constraints are included. In his algorithm a maximal set of independent modules (that do not communicate with one another) is first found and these modules will be assigned after other modules. The lower bound is estimated under the assumption that there is no communication among free modules. Thus it is unnecessary to expand a node if the free modules on the node are independent of one another. Unfortunately, this is not true if constraints are taken into consideration. In this paper, using a different branch-and-bound algorithm, we have solved the module assignment problem with constraints. The experimental results show that the algorithm has an accurate estimation of the lower bound.

Acknowledgment

The authors wish to thank the Editor and anonymous referees for their helpful suggestions and comments.

Appendix: An example

Suppose that there are five program modules $M_1, M_2, \ldots, M_5$ and their interconnection pattern is shown in Fig. 2, where M_i–M_j means that M_i communicates with M_j during execution. There are three processors P_1, P_2 and P_3 available. The execution costs and communication costs are shown in Table 3 and Table 4 respectively. There are storage constraints and load constraints imposed on processors. The values of *STOR* and *LOAD* for each program module are shown in Table 5 and the values of *SUB* and *LUB* for each processor are shown in Table 6. M_1 and M_2 are restricted to the same processor, M_1 is restricted to P_1 and P_2, and M_4 is restricted to P_3. To have the illustration simpler, we assume that an initial feasible solution $M_1 \rightarrow P_1$, $M_2 - P_1$, $M_3 \rightarrow P_2$, $M_4 \rightarrow P_3$, $M_5 \rightarrow P_1$, whose cost is 110, is obtained by heuristics. The branch-and-bound tree is shown in Fig. 3, where "$M_i \rightarrow P_j$" in each node means "assign program module M_i to processor P_j". But, "$M_i \rightarrow P_j$" on an edge represents the restriction that program module M_i is restricted to processor P_j and "$M_i \nrightarrow P_j$" on an edge represents the restriction that program module M_i is not allowed to be assigned to processor P_j. The numbers in parentheses represent the

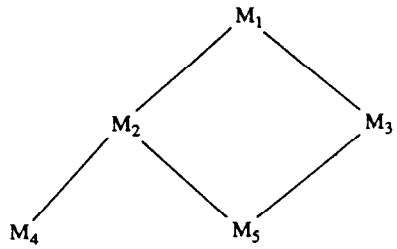

Fig. 2. Interconnection pattern of $M_1, M_2, \ldots, M_5$.

Table 3
Execution costs

	M_1	M_2	M_3	M_4	M_5
P_1	10	10	21	∞	9
P_2	6	14	9	∞	12
P_3	∞	15	13	22	17

Table 4
Communication costs

M_2

		P_1	P_2	P_3
	P_1	0	5	7
M_1	P_2	4	0	8
	P_3	11	10	0

M_3

		P_1	P_2	P_3
	P_1	0	16	24
M_1	P_2	19	0	23
	P_3	22	23	0

M_4

		P_1	P_2	P_3
	P_1	0	29	23
M_2	P_2	27	0	21
	P_3	23	22	0

M_5

		P_1	P_2	P_3
	P_1	0	18	19
M_2	P_2	12	0	16
	P_3	20	18	0

M_5

		P_1	P_2	P_3
	P_1	0	19	20
M_3	P_2	11	0	13
	P_3	14	15	0

Table 5
The values of *STOR* and *LOAD*

	M_1	M_2	M_3	M_4	M_5
STOR	16	9	18	24	12
LOAD	13	8	11	16	9

Table 6
The values of *SUB* and *LUB*

	P_1	P_2	P_3
SUB	40	50	40
LUB	35	40	30

sequence in which the nodes are generated. The values of CC and EC are also shown in Fig. 3. Tracing Fig. 3, we can find that the optimal solution is $M_1 \rightarrow P_2$, $M_2 \rightarrow P_2$, $M_3 \rightarrow P_2$, $M_4 \rightarrow P_3$, $M_5 \rightarrow P_1$, the cost of which is 104.

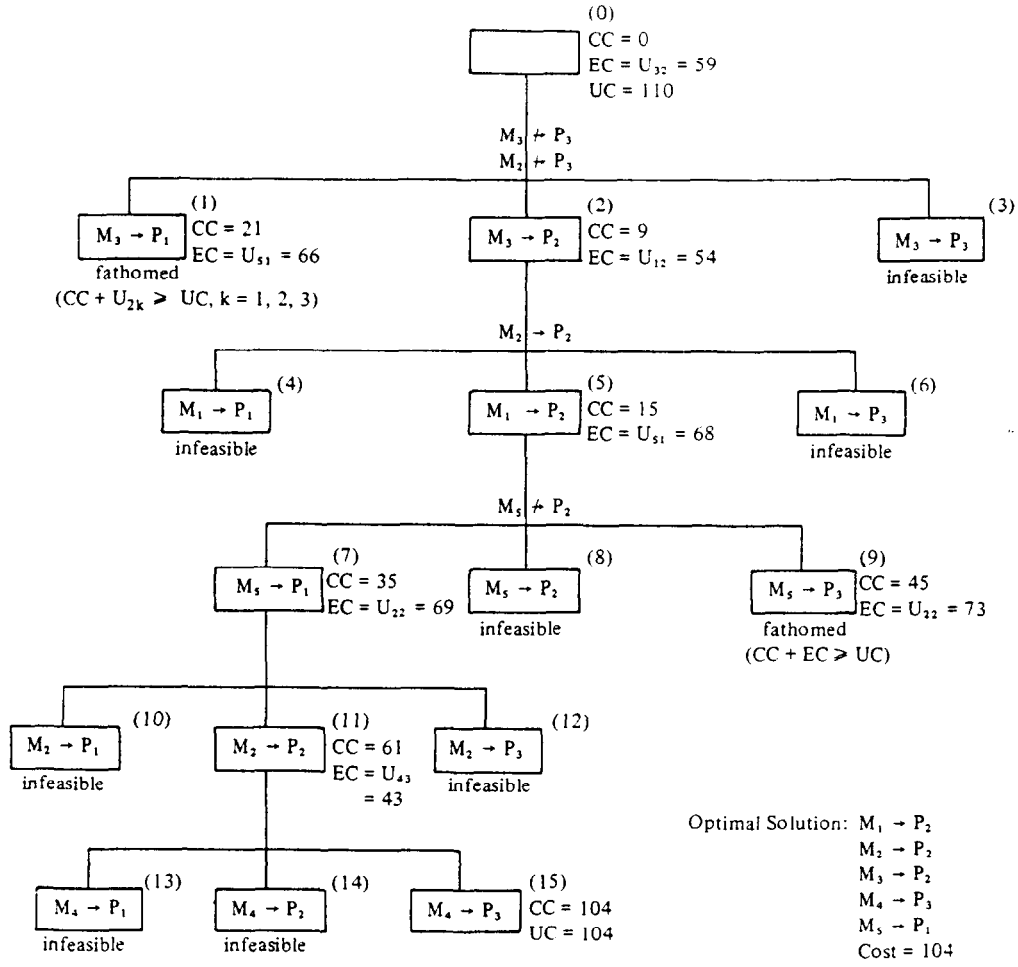

Fig. 3. The branch-and-bound tree.

References

[1] R.K. Arora and S.P. Rana, On module assignment in two-processor distributed systems, *Inform. Process. Lett.* **9** (1979) 113–117.

[2] R.K. Arora and S.P. Rana, Analysis of the module assignment problem in distributed computing systems with limited storage, *Inform. Process. Lett.* **10** (1980) 111–115.

[3] S.H. Bokhari, Dual processor scheduling with dynamic reassignment, *IEEE Trans. Software Eng.* **SE-5** (1979) 341–349.

[4] S.H. Bokhari, A shortest tree algorithm for optimal assignments across space and time in a distributed processor system, *IEEE Trans. Software Eng.* **SE-7** (1981) 583–589.

[5] T.L. Casavant and J.G. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, *IEEE Trans. Software Eng.* **SE-14** (1988) 141–154.

[6] W.W. Chu and L.M.-T. Lan, Task allocation and precedence relations for distributed real-time systems, *IEEE Trans. Comput.* **C-36** (1987) 667–679.

[7] C. Gao, J.W.S. Liu and M. Railey, Load balancing algorithms in homogeneous distributed systems, In: *Proc. 1984 Internat. Conf. on Parallel Processing* (1984) 302–306.

[8] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms* (Computer Science Press, Potomac, MD, 1978).

[9] B. Indurkhya, H.S. Stone and X.-C. Lu, Optimal partitioning of randomly generated distributed programs, *IEEE Trans. Software Eng.* **SE-12** (1986) 483–495.

[10] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, *IEEE Trans. Comput.* **C-37** (1988) 1384–1397.

[11] P.R. Ma, E.Y.S. Lee and M. Tsuchiya, A task allocation

model for distributed computing systems, *IEEE Trans. Comput.* **C-31** (1982) 41–47.

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

[13] C.C. Shen and W.H. Tsai, A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion, *IEEE Trans. Comput.* **C-34** (1985) 197–203.

[14] J.B. Sinclair, Efficient computation of optimal assignments for distributed tasks, *J. Parallel Distributed Comput.* **4** (1987) 342–362.

[15] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Eng.* **SE-3** (1977) 85–93.

[16] H.S. Stone and S.H. Bokhari, Control of distributed processes, *Computer* **11** (1978) 97–106.