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 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) \( T(M_i, M_j, P_k, P_l) \) is the communication cost that is incurred by program modules \( M_i \) and \( M_j \) when they are assigned to processor \( P_k \) and \( P_l \) 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 \( \text{SUB}(P_j) \) denote the storage limit for processor \( P_j \), and \( \text{STOR}(M_i) \) denote the amount of storage occupied by program module \( M_i \). If \( M_{r_1}, M_{r_2}, \ldots, M_{r_k} \) are all assigned to \( P_r \), then \( \text{STOR}(M_{r_1}) + \text{STOR}(M_{r_2}) + \cdots + \text{STOR}(M_{r_k}) \) must not exceed \( \text{SUB}(P_r) \).

2. **Load constraint:** The available computational load provided by a processor has an upper bound. Let \( \text{LUB}(P_j) \) denote the computational load upper bound for processor \( P_j \), and \( \text{LOAD}(M_i) \) denote the computational load required by program module \( M_i \). If \( M_{r_1}, M_{r_2}, \ldots, M_{r_k} \) are all assigned to \( P_r \), then \( \text{LOAD}(M_{r_1}) + \text{LOAD}(M_{r_2}) + \cdots + \text{LOAD}(M_{r_k}) \) must not exceed \( \text{LUB}(P_r) \).

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 \( \psi \) be a mapping from \( M \) to \( P \), i.e., \( \psi: M \rightarrow P \). The problem is to minimize

\[
\sum_{i=1}^{m} E(M_i, \psi(M_i)) + \sum_{i=1}^{m} \sum_{j=i+1}^{m} T(M_i, M_j, \psi(M_i), \psi(M_j))
\]

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 \( (m\text{th}) \) 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 \( P_i \). Let \( S \) denote the set of program modules that are included in the corresponding partial solution. \( U_{ij} \) is estimated as follows:

\[
U_{ij} = E(M_i, P_j) + \sum_{M_s \in S} T(M_s, M_i, P_j, P_{\phi(z)}) \\
+ \sum_{M_s \notin S \cup \{M_i\}} \min \left\{ E(M_s, P_j) + \sum_{M_t \in S} T(M_t, M_s, P_j, P_{\phi(z)}) \\
+ T(M_t, M_i, P_j, P_j) \mid r \in \{1, 2, \ldots, n\} \right\} \text{ where } M_i \rightarrow P_i \text{ is feasible}
\]

where \( P_{\phi(\epsilon)} \) and \( P_{\phi(z)} \) are the respective processors to which \( M_s \) and \( M_i \) were assigned. \( U_{ij} \) is set to infinity if \( M_i \rightarrow P_j \) is infeasible or \( M_i \rightarrow P_j \) is infeasible for all processors \( P_j \). 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:

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

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 initially and is updated to be \( \min \{ UC, CC \} \) whenever a feasible node at the lowest level is reached. If a node satisfies \( CC + EC \geq UC \), then it is fathomed, since further branching from it will not lead to better solutions. If a node satisfies \( CC + U_{ij} \geq UC \) 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 \( M_i \rightarrow P_j \)). Thus a node can be fathomed if it satisfies \( CC + U_{ij} \geq UC \) for some free program module \( M_i \) and all processors \( P_j \). Similarly, if a node satisfies \( CC + D_{ij} \geq UC \) for some free program module \( M_i \) and some processor \( P_j \), then \( M_i \) is restricted to \( P_j \) (denoted by \( M_i \rightarrow P_j \)). These restrictions will cause more nodes to be fathomed.
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 \( U_{i,j}, D_{i,j}, EC \), and \( CC \) for root node;
3. set \( UC \) to be infinity;
4. while live-node-list is not empty do
   begin
   choose a node \( \alpha \) with minimum value of \( CC + EC \) from live-node-list;
   if \( \alpha \) has \( CC + EC \geq UC \) or \( CC + U_{i,j} \geq UC \) for some free \( M_i \) and all \( P_j, j = 1, 2, \ldots, n \) then
   remove \( \alpha \) from live-node-list
   else begin
   if \( CC + D_{i,j} \not\geq UC \) for some free \( M_i \) and some \( P_j \) then a restriction “\( M_i \rightarrow P_j \)” is added;
   branch \( \alpha \) (assign \( M_i \) that satisfies \( EC = U_{i,j} \) for some \( P_j \) to all processors);
   for each (assume \( \beta \)) of newly generated nodes do
   if \( \beta \) is feasible then begin
   compute \( U_{i,j}, D_{i,j}, EC \), and \( CC \);
   if \( \beta \) is at the lowest level
   then if \( CC < UC \) then replace \( UC \) with \( CC \)
   else if \( CC + EC < UC \) then insert \( \beta \) into live-node-list
   end;
   remove \( \alpha \) from live-node-list
   end
   end;
5. 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

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>[0.10] p=0.2</th>
<th>[0.10] p=0.5</th>
<th>[0.10] p=0.8</th>
<th>[0.50] p=0.2</th>
<th>[0.50] p=0.5</th>
<th>[0.50] p=0.8</th>
<th>[0.100] p=0.2</th>
<th>[0.100] p=0.5</th>
<th>[0.100] p=0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>0.96047</td>
<td>0.94724</td>
<td>0.94011</td>
<td>0.95107</td>
<td>0.93072</td>
<td>0.91115</td>
<td>0.92915</td>
<td>0.92681</td>
<td>0.95655</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0.97087</td>
<td>0.95992</td>
<td>0.94584</td>
<td>0.93919</td>
<td>0.91964</td>
<td>0.94701</td>
<td>0.93317</td>
<td>0.94819</td>
<td>0.97204</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.98114</td>
<td>0.96668</td>
<td>0.96247</td>
<td>0.92701</td>
<td>0.95789</td>
<td>0.97743</td>
<td>0.95238</td>
<td>0.95844</td>
<td>0.98114</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0.98339</td>
<td>0.97152</td>
<td>0.96833</td>
<td>0.96228</td>
<td>0.97015</td>
<td>0.98432</td>
<td>0.94681</td>
<td>0.95745</td>
<td>0.99291</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0.99313</td>
<td>0.98723</td>
<td>0.98151</td>
<td>0.98244</td>
<td>0.97384</td>
<td>0.98986</td>
<td>0.96375</td>
<td>0.96234</td>
<td>0.99499</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>0.98885</td>
<td>0.96831</td>
<td>0.96432</td>
<td>0.97530</td>
<td>0.96292</td>
<td>0.96671</td>
<td>0.96027</td>
<td>0.95288</td>
<td>0.97937</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0.99552</td>
<td>0.98761</td>
<td>0.96805</td>
<td>0.97910</td>
<td>0.98860</td>
<td>0.99511</td>
<td>0.98453</td>
<td>0.97905</td>
<td>0.98622</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.99414</td>
<td>0.98428</td>
<td>0.98315</td>
<td>0.98531</td>
<td>0.99272</td>
<td>0.99681</td>
<td>0.98669</td>
<td>0.99960</td>
<td>0.99694</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.99732</td>
<td>0.98720</td>
<td>0.97985</td>
<td>0.98997</td>
<td>0.99376</td>
<td>0.99841</td>
<td>0.96377</td>
<td>0.98771</td>
<td>0.99956</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>0.99733</td>
<td>0.98429</td>
<td>0.96527</td>
<td>0.99602</td>
<td>0.99685</td>
<td>0.99922</td>
<td>0.99282</td>
<td>0.99947</td>
<td>0.99968</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>0.99713</td>
<td>0.99446</td>
<td>0.99496</td>
<td>0.99862</td>
<td>0.99847</td>
<td>0.99935</td>
<td>0.98794</td>
<td>0.99950</td>
<td>0.99980</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>0.99882</td>
<td>0.99207</td>
<td>0.99998</td>
<td>0.99331</td>
<td>0.99868</td>
<td>0.99983</td>
<td>0.97941</td>
<td>0.99961</td>
<td>0.99992</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.99888</td>
<td>0.99730</td>
<td>0.99420</td>
<td>0.99553</td>
<td>0.99934</td>
<td>0.99998</td>
<td>0.99189</td>
<td>0.99981</td>
<td>0.99995</td>
</tr>
</tbody>
</table>

The saving rate is defined to be \(1 - \frac{N(n-1)}{(n^{m+1}-1)}\), where \(N\) is the number of nodes that are actually generated by the branch-and-bound algorithm and \(\frac{(n^{m+1}-1)}{(n-1)}\) 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

\[
LUB = \frac{3}{5} \cdot \sum_{i=1}^{m} LOAD(M_i).
\]

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

<table>
<thead>
<tr>
<th>Optimal cost</th>
<th>Random assignment method</th>
<th></th>
<th>Least cost</th>
<th>Average cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Number of successes</td>
<td>Greatest cost</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 4</td>
<td>172</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>161</td>
<td>34</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td>108</td>
<td>21</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>183</td>
<td>37</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 5</td>
<td>119</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>173</td>
<td>32</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td></td>
<td>193</td>
<td>36</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 6</td>
<td>130</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>147</td>
<td>32</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td></td>
<td>164</td>
<td>34</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 7</td>
<td>110</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>260</td>
<td>35</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td></td>
<td>175</td>
<td>22</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 8</td>
<td>150</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>287</td>
<td>31</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td></td>
<td>260</td>
<td>38</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 9</td>
<td>272</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>222</td>
<td>28</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td></td>
<td>253</td>
<td>39</td>
<td>253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>106</td>
<td>31</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 10</td>
<td>296</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>73</td>
<td>24</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220</td>
<td>32</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m = 10</td>
<td>339</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>375</td>
<td>33</td>
<td>375</td>
</tr>
</tbody>
</table>
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,

\[
C_{ij} = \max_{t=1, t \neq i} \left\{ T(M_i, M_t, P_j, P_r) \mid r = 1, 2, \ldots, n \right\}.
\]

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

\[
\Delta' - \Delta^* = E(M_i, P_j) + \sum_{r=1, r \neq i}^m T(M_i, M_r, P_j, \psi'(M_r)) - E(M_i, P_v) - \sum_{r=1, r \neq i}^m T(M_i, M_r, P_v, \psi^*(M_r)) \leq E(M_i, P_j) - E(M_i, P_v) + \sum_{r=1, r \neq i}^m T(M_i, M_r, P_j, \psi(M_r)) \leq E(M_i, P_j) - E(M_i, P_v) + C_{ij} \leq 0.
\]

So, \( \psi' \) is an optimal assignment. \( \square \)

Suppose that \( M_i \) only communicates with \( M_u. \) Let \( SS_{\text{max}} \) 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 \( SR_{\text{min}} \) 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,

\[
SS_{\text{max}} = \max \left\{ E(M_i, P_j) + E(M_u, P_j) \mid j = 1, 2, \ldots, n \right\},
\]

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

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

Proof. Suppose that $\psi^*$ 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 $\psi'$ from $\psi^*$ by changing the assignment of $M_i$ from $P_j$ to $P_v$. Let $\Delta^*$ be the total cost of $\psi^*$ and $\Delta'$ be the total cost of $\psi'$. We have

$$\Delta' - \Delta^* = E(M_i, P_v) - [E(M_i, P_j) + T(M_i, M_u, P_j, P_v)]$$

$$= [E(M_i, P_v) + E(M_u, P_v)] - [E(M_i, P_j) + E(M_u, P_v) + T(M_i, M_u, P_j, P_v)]$$

$$\leq SS_{\text{max}} - SR_{\text{min}} \leq 0$$

So, $\psi'$ is an optimal assignment. $\square$

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. $M_i$ and $M_2$ are restricted to the same processor, $M_3$ 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 $M_1 \rightarrow P_1, M_2 \rightarrow P_1, M_3 \rightarrow P_2, M_4 \rightarrow P_3, M_5 \rightarrow P_1$, 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 "$M_i \leftrightarrow P_j$" 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 $M_1$, $M_2$, ..., $M_5$. 

Table 3
Execution costs

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>10</td>
<td>10</td>
<td>21</td>
<td>$\infty$</td>
<td>9</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6</td>
<td>14</td>
<td>9</td>
<td>$\infty$</td>
<td>12</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$\infty$</td>
<td>15</td>
<td>13</td>
<td>22</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 4
Communication costs

<table>
<thead>
<tr>
<th></th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$P_1$</td>
<td>$P_1$</td>
<td>$P_1$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$P_2$</td>
<td>$P_2$</td>
<td>$P_2$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$P_3$</td>
<td>$P_3$</td>
<td>$P_3$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$STOR$</td>
<td>16</td>
<td>9</td>
<td>18</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>$LOAD$</td>
<td>13</td>
<td>8</td>
<td>11</td>
<td>16</td>
<td>9</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SUB$</td>
<td>40</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>$LUB$</td>
<td>35</td>
<td>40</td>
<td>30</td>
</tr>
</tbody>
</table>

The values of $STOR$ and $LOAD$ are shown in Table 5. Tracing Fig. 3, we can find that the optimal solution is $M_1 \to P_2$, $M_2 \to P_2$, $M_3 \to P_2$, $M_4 \to P_3$, $M_5 \to P_1$, the cost of which is 104.
Fig. 3. The branch-and-bound tree.

References


