

## Grouping components in printed circuit board assembly with limited component staging capacity and single card setup: problem characteristics and solution procedures

M. S. DASKIN†, O. MAIMON‡, A. SHTUB‡ and D. BRAHA§\*

The problem of grouping Printed Circuit Board (PCB) components to minimize the total component and PCB loading cost subject to a capacity constraint on the number of types of components per group is formulated as an integer linear programming problem. The problem is shown to be NP-complete. Characteristics of the solution are outlined and a heuristic algorithm is discussed. For the case in which it is optimal to load each PCB exactly once, the solution characteristics can be used to obtain a lower bound on the objective function for any set of constraints on pairs of PCBs that must be produced using the same group of components. The bounds and the heuristic procedure are used to develop a branch and bound algorithm. Computational results are given for four test problems derived from industrial contexts.

### 1. Introduction, problem statement and related literature

Printed circuit boards (PCB) assembly involves mainly the insertion and soldering of electrical components into printed circuit boards. Significant cost savings have resulted from the automation of the process of PCB assembly (Boothroyd 1992). Rockwell and Wilhelm (1990) summarized the production of PCBs in some detail.

In this work we consider the short range production planning of the automatic assembly of PCBs. The focus in short-range (operational) planning is mostly on the sequencing problem. This is because a clear trade-off exists between the sequence that completes each PCB type once it is loaded on to the machine (minimizing the number of times each PCB is loaded on to the machine), and a sequence that simultaneously loads all PCBs using the same group of components loaded onto the machines (minimizing the number of times each component type is loaded onto the machine). For a thorough review on automated process planning for PCB assembly and list of references, related to the many possible strategies for managing PCB assembly resources, see McGinnis, *et al.* (1992).

We assume that we are given a number of different PCB types that are to be produced. Associated with each PCB type is a list of the component types that must be inserted into the PCB. Underlying the problem formulation are the following assumptions: (1) components are to be inserted into the PCBs by a machine that has a finite capacity for different types of components but that effectively has an

---

Revision received December 15, 1995.

† Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, USA.

‡ Department of Industrial Engineering, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel.

§ Manufacturing Engineering Department, Boston University, Boston, MA 02215, USA.

\* To whom correspondence should be addressed at Department of Industrial Engineering and Management, Ben Gurion University, Beer-Sheva 84105, Israel.

infinite capacity for components of any specific type. In that case, the machine capacity is not determined optimally; (2) costs are incurred each time a different type of PCB is loaded on to the machine as well as when a component is included in a group (i.e. removing a given component for one setup and then staging it again for some subsequent setup); and (3) permanent components (i.e. those that remain on the machine at all times) are not treated by the solution methods.

In the general case, we allow a PCB to be loaded multiple times (one per group) and we allow components to be included in multiple groups. However, the proposed solution methods will focus on a restricted class of problems in which it is feasible and optimal to load each PCB only once. In that case, all of the components required by the PCB must clearly be included in the group in which the PCB is to be produced.

In this paper, the 'partition and repeat' strategy (see McGinnis, *et al.* 1992) is used for managing PCB assembly resources. Following McGinnis *et al.* (1992), a partition and repeat strategy (PAR) partitions the components required by the family into subsets such that the group has enough staging capacity for each subset. The group is configured to run each subset in turn, requiring the accumulation of all the partially completed PCBs in the family. The advantage of the PAR strategy is that it requires relatively few setups; the disadvantage is that it requires every PCB in the family to be accumulated as partially completed work in process.

The problem we address is: A set of PCBs are to be produced on a single machine with limited component staging capacity. The objective is to minimize a weighted sum of the total number of times at which PCB types are switched; and the total of the subset cardinalities. When it is both feasible and optimal to load each PCB onto the machine only once (the class of problems actually treated), the PAR strategy can thus be used for PCBs, so the problem is to determine the subset of PCBs in each group.

Considerable research has been devoted to the problem of finding appropriate groups for cellular manufacturing (Singh 1993, Shafer and Meredith 1990). The problem is often cast in terms of grouping machines into cells such that the production of all components can be accomplished by machines in a single cell. Often this is not possible unless machines can be included in more than one cell or unless components are moved between cells during production. In terms of our problem, components in the broader 'group technology' (GT) literature are analogous to PCBs and machines are analogous to electrical components (which we often refer to simply as components). Having several copies of the same machine (i.e. including a machine in more than one cell) is analogous to allowing an electrical component to be included in multiple groups. Similarly, moving a component between cells during production is analogous to producing (and hence loading) a PCB using multiple groups of electrical components. A key facet of our problem that is often ignored in the GT literature is that there is a limit on the number of different electrical component types that can be included in a group. This is analogous to having a limit on the number of different machines that can be included in a cell. Harhalakis *et al.* (1990) have recently proposed a heuristic approach to cell formation that reflects this constraint.

Several approaches exist to group the machines into cells, such as (1) the application of clustering techniques to a part-machine matrix (King 1980a, 1980b; Chandrasekharan and Rajagopalan 1986, Maimon and Shtub 1991, amongst others), (2) the use of optimization-based approaches (e.g. Kusiak 1987, Kumar

*et al.* 1986, Shtub 1989), (3) graph theoretic approaches (e.g. Rajagopalan and Batra 1975, Vannelli and Kumar 1986, Askin and Chiu 1990), (4) simulated annealing (Harhalakis *et al.* 1990, Liu and Wu 1993), genetic algorithms (Pierreval and Plaquin 1994), neural networks (Kaparathi *et al.* 1993, Chung and Kusiak 1994) as well as the use of learning methods to minimize the traffic between cells (Chu 1993). A literature review summarizing these approaches is given in Singh (1993).

The remainder of the paper is organized as follows. In § 2 we formulate the problem as an interger linear programming problem. Unlike most of the diagonalization and similarity based approaches, the number of different types of electrical components that can be included in a group (or equivalently the number of machines that can be in a cell) are limited. In § 3 we provide conditions under which it is optimal to load each PCB exactly once and show that the problem is NP-complete. In § 4 we outline a heuristic solution procedure. In § 5, we use characteristics of the solution to the problem when each PCB is to be loaded only once to derive bounds on the objective function. Using these bounds we develop a branch and bound algorithm that is shown to be effective at optimally solving small to moderate sized instances of the problem. In § 6 we summarize our computational experience with the branch and bound algorithm. Finally, conclusions and recommendations for future research are outlined in § 7.

## 2. Mathematical formulation

To formulate the problem of grouping components and PCBs, we define the following indices:

- $i$  index of component types ( $i = 1, \dots, I$ )
- $j$  index of PCBs ( $j = 1, \dots, J$ )
- $g$  index of groups ( $g = 1, \dots, G$ )

We define the following inputs:

- $s_i$  the loading cost of component type  $i$
- $S_j$  the loading cost of PCB  $j$
- $L$  the maximum number of component types allowed in a group
- $a_{ij}$  1, if component type  $i$  is needed by PCB  $j$ ; 0 otherwise
- $N_j$  the set of indices of component types  $i$  required by PCB  $j = \{i | a_{ij} = 1\}$
- $O_j$  the set of indices of components types  $i$  required *only* by PCB  $j = \{i | a_{ij} = 1 \text{ and } a_{ij'} = 0 \forall j' \neq j\}$
- $C_j$  the set of indices of component types  $i$  required by PCB  $j$  *and* at least one other PCB  
 $\{i | a_{ij} = 1 \text{ and } \exists j' \neq j \text{ such at } a_{ij'} = 1\}$

$S_j$  is the set up cost associated with producing a batch of PCBs of type  $j$ . This cost will be incurred for each group used in the production of PCB type  $j$ . PCB types with identical component requirements may be included separately to allow the model formulated below to capture the set up costs of each PCB type. Similarly, the component cost,  $s_i$  will be incurred for each group in which component  $i$  is included. We assume that the costs  $s_i$  and  $S_j$  are strictly positive.

We define the following decision variables:

$$X_{ijg} = \begin{cases} 1, & \text{if component type } i \text{ of PCB } j \text{ is in group } g; \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{jg} = \begin{cases} 1, & \text{if PCB } j \text{ is produced entirely (or in part) in group } g; \\ 0, & \text{otherwise} \end{cases}$$

$$Z_{ig} = \begin{cases} 1, & \text{if component type } i \text{ is in group } g; \\ 0, & \text{otherwise} \end{cases}$$

Note that if component type  $i$  is in group  $g$ , it can be used in the production of any PCB  $j$  that requires component type  $i$ . With this notation, the *PCB-grouping* problem may be formulated as follows:

$$\text{Minimize } \sum_i s_i \sum_g Z_{ig} + \sum_j S_j \sum_g Y_{jg} \quad (1)$$

Subject to:

$$\sum_g X_{ijg} = 1 \quad \forall i \in N_j, \forall j \quad (2)$$

$$Z_{ig} \geq X_{ijg} \quad \forall i \in N_j, \forall j \forall g \quad (3)$$

$$Y_{jg} \geq X_{ijg} \quad \forall i \in N_j, \forall j \forall g \quad (4)$$

$$\sum_i Z_{ig} \leq L \quad \forall g \quad (5)$$

$$X_{ijg} \in \{0, 1\} \quad \forall i \in N_j, \forall j \forall g \quad (6)$$

$$Y_{jg} \geq 0 \quad \forall j \forall g \quad (7)$$

$$Z_{ig} \geq 0 \quad \forall i \forall g \quad (8)$$

The objective function (1) is for minimizing the sum of the component and PCB loading costs. We assume that the component loading cost is incurred for each component type in each group (whether or not the component is in any other group). Thus, even if a component is in two groups that are to be loaded on the machine one after the other, the objective function counts the component loading cost a second time. Constraint (2) stipulates that each component-PCB combination be assigned to exactly one group. Note that it does not require that each component be in only one group. Constraint (3) states that if component  $i$  of PCB  $j$  is in group  $g$  ( $X_{ijg} = 1$ ), then component  $i$  must be assigned to group  $g$  ( $Z_{ig} = 1$ ). Similarly, constraint (4) states that if component  $i$  of PCB  $j$  is in group  $g$ , then PCB  $j$  must be loaded with group  $g$  ( $Y_{jg} = 1$ ). Constraint (5) is the capacity constraint for the number of component types permitted in a group. Constraint (6) is the integrality constraint on the assignment variables  $X_{ijg}$ . Because of constraints (3) and (4), the fact that the objective function is a minimization with positive coefficients, and the integrality constraint on the assignment variables  $X_{ijg}$ , the decision variables  $Y_{jg}$  and  $Z_{ig}$  need only be constrained to be non-negative as shown in equations (7) and (8). To prove this result, let us consider the following cases: (a) If  $X_{ijg} = 1$  for some  $j$ , then  $Z_{ig} = 1$  is a feasible solution that minimizes the objective function (1); (b) If  $X_{ijg} = 0$  for every  $j$ , then  $Z_{ig} = 0$  is a feasible solution that minimizes the objective function (1). Similar arguments hold for  $Y_{jg}$ .

Formulation (1)–(8) allows both PCBs and components to be loaded multiple times if doing so will minimize the total loading cost. Maimon and Shtub (1991) identified this as a type 4 problem where a type 1 problem constrains each PCB and each component to be loaded only once; a type 2 problem constrains each component to be loaded only once; and a type 3 problem constrains each PCB to be loaded only once. Maimon and Shtub (1991) provide a non-linear integer programming formulation of the general (type 4) problem. By introducing the variable  $X_{ijg}$ , we obtain a linear integer programming formulation.

To obtain a type 3 formulation, we append the following constraint to formulation (1)–(8):

$$\sum_g Y_{jg} = 1 \quad \forall j \quad (9)$$

Similarly, to obtain a type 2 formulation, we add the following constraint to formulation (1)–(8):

$$\sum_g Z_{ig} = 1 \quad \forall i \quad (10)$$

Finally, to obtain a type 1 formulation both constraints (9) and (10) should be added to formulation (1)–(8).

Other variations on the basic model may also be formulated:

- In some applications that we observed in industry, a set of permanent components are identified. These components remain on the machine at all times; that is, they are not changed when groups are changed. These components are (presumably) the components that are required by many PCBs. We assume that no cost is incurred in loading the permanent components. To identify simultaneously a set of permanent components and the components to include in each of the groups, we define group 0 as the group of permanent components.
- Formulation (1)–(8) does not force or induce small groups to be combined. However, two small groups may be identified which have no components in common and which could be merged into a single group without violating the capacity limit on the number of different components in a group. To model such problems, we can introduce a cost of using an additional group.

In what follows, when we refer to the PCB-grouping problem, we are referring to formulation (1)–(8). We will be primarily interested in the solution to the problem when each PCB is loaded exactly once (i.e. a type 3 problem).

### 3. Properties of the PCB-grouping problem

In this section we outline a number of properties of the solution of the PCB-grouping problem formulated above. First, (1) if the machine capacity,  $L$ , is greater than or equal to the number of components needed by each PCB (i.e.  $L \geq \lceil N_j \rceil \forall j$ ) and (2) if all of the PCB loading costs,  $S_j$  are very large relative to the component loading costs,  $s_i$ , then it will be optimal to load each PCB only once (i.e. a type 3 solution will be generated). That is, for sufficiently large values of  $S_j$  and  $L$ , the optimal solution to (1)–(8) will automatically satisfy constraint (9). This is formalized in the following theorem.

*Theorem 1:* If  $|N_j| \leq L$  and  $S_j > \sum_{i \in V_j} s_i - \sum_{i \in \mathcal{O}_j} s_i = \sum_{i \in C_j} s_i$ , then PCB  $j$  should be loaded only once in an optimal solution.

*Proof:* Since  $|N_j| \leq L$  it is feasible to load all components required by PCB  $j$  in the same group. Thus, an upper bound on the cost of loading the components required by PCB  $j$  and the PCB loading cost is  $S_j + \sum_{i \in V_j} s_i$ . If PCB  $j$  is loaded multiple times, then a lower bound on the production cost of PCB  $j$  is  $2S_j + \sum_{i \in \mathcal{O}_j} s_i$ . The validity of this bound results from the fact that we are counting the loading cost of PCB  $j$ ,  $S_j$ , twice and are adding to that *only* the component loading costs of the components used by PCB  $j$  only. Thus, if  $2S_j + \sum_{i \in \mathcal{O}_j} s_i > S_j + \sum_{i \in V_j} s_i$  or  $S_j > \sum_{i \in V_j} s_i - \sum_{i \in \mathcal{O}_j} s_i = \sum_{i \in C_j} s_i$  then it is cheaper to load PCB  $j$  only once.  $\square$

The main result in this section concerns the computational complexity of the PCB-grouping problem. Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources—such as computation time, space and hardware size—required to solve problems via digital algorithms. A *problem* is a collection of *instances* that share a mathematical form but differ in size and in the values of numerical constants in the problem form. In general we convert optimization problems to *decision problems* by posing the question of whether there is a feasible solution to a given problem with objective function value equal or superior to a specified threshold. The notion of ‘easy to verify’ but ‘not necessarily easy to solve’ decision problems is at the heart of the Class *NP*. Specifically, *NP* includes all those decision problems that could be polynomial-time solved if the right (polynomial-length) ‘clue’ or ‘guess’ were appended to the problem input string. An important subclass of *NP* problems are referred to as *NP-complete* or non-deterministic polynomial time-complete problem (Garey and Johnson 1979). The CPU time required to solve an *NP-complete* problem, based on known algorithms, grows exponentially with the ‘size’ of the problem. There exists no polynomial time transformations for *NP-complete* problems, nor are there any polynomial time algorithms capable of solving any *NP* problems. The potential to solve *NP* and *NP-complete* problems depends on the availability of certain heuristics. One problem *polynomially reduced* to another if a polynomially bounded number of calls to an algorithm for the second will always solve the first. A problem  $\Omega$  is shown to be *NP-complete* by polynomially reducing another already known *NP-complete* problem to  $\Omega$ .

We now show that the PCB-grouping problem is *NP-complete* in the strong sense. We do this by showing that the *3-Partition* problem, which is *NP-complete* in the strong sense (Garey and Johnson 1979, pp. 96–100) polynomially reduces to a special case of the PCB-grouping problem. The *3-Partition* decision problem is defined as follows:

**INSTANCE:** An integer  $B$ ,  $3m$  integer numbers of  $d_i$  satisfying (i)  $B/4 < d_i < B/2$  and (ii)  $\sum_i d_i = mB$ .

**QUESTION:** Does there exist a partition of the  $d_i$  into  $m$  disjoint sets  $P_1, P_2, \dots, P_m$  such that  $\sum_{i \in P_g} d_i = B$  for each set  $P_g$ ?

If the answer to the *3-Partition* decision problem is yes, each set  $P_g$  will contain exactly 3 elements (because of the bounds (i) on the values  $d_i$ ). We can define the PCB-grouping decision problem by simply asking whether or not a solution to the PCB-grouping optimization problem exists with a value less than or equal to some

input constant  $Z$  (recall that this problem is computationally equivalent to the original problem).

*Theorem 2:* The PCB-grouping problem is NP-complete in the strong sense.

*Proof:* The proof is given in appendix A.

From the proof it can be shown that the PCB-grouping problem remains NP-complete even (1) when no PCB is produced using more than a single group (i.e. constraint (9) is binding) and/or (2) when each group is constrained to have 3 or fewer PCBs and/or (3) when all PCB loading costs are identical and all component loading costs are identical.

We note that if each group is constrained to have 2 or fewer PCBs and no PCB is produced using more than a single group, the problem can be solved in polynomial time using a minimum weighted matching algorithm. The graph for this problem is constructed as follows (an example is presented in fig. 1).

*Example Component/PCB Requirement Matrix (with staging capacity,  $L=4$ ):*

<u>COMPONENT</u>	<u>PCB</u>			
	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>
1	1	1	0	0
2	1	0	1	0
3	0	1	0	0
4	1	0	1	0
5	0	0	1	1

*The Associated Graph Representation:*

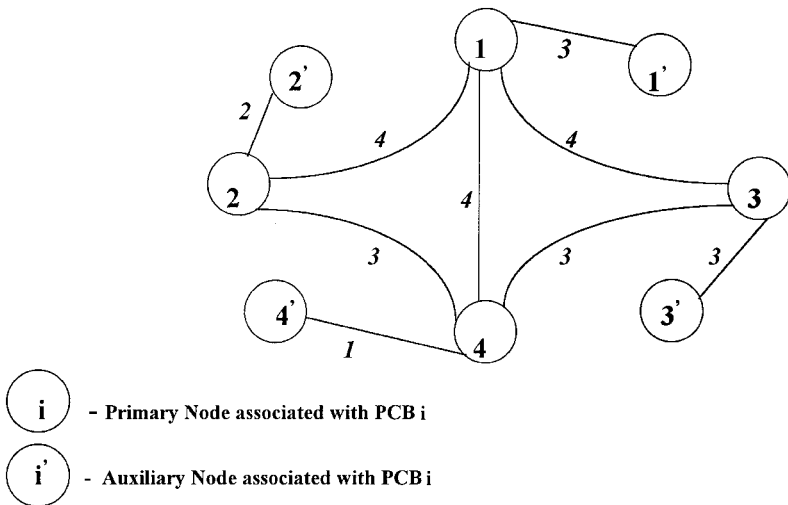


Figure 1. Transforming a PCB-grouping problem into a minimum weighted matching.

We associate a primary node with each PCB. A link connects each pair of primary nodes associated with PCBs that can be produced in the same group (i.e. PCBs whose combined component requirements do not exceed the capacity of the machine). The cost of each such link is the total component loading cost of the components that are required by the two PCBs. In addition, we create an auxiliary node for each PCB. A link is added to the graph between the auxiliary and primary nodes for each PCB. The cost associated with such links is simply the total component loading cost of the PCB's components. (Such links allow for the possibility of having groups that can produce only one PCB.) The solution to the minimum weighted matching problem (which can be obtained in  $O(J^3)$  time (Papadimitriou and Steiglitz 1982) identified the groups that should be formed to allow production of pairs of PCBs as well as the groups that should be formed to produce only a single PCB.

In the remainder of this paper, we assume that  $S_j > \sum_{i \in C_j} s_i \forall j$  and that  $|\mathbb{N}_j| \leq L \forall j$  so that each PCB will be loaded exactly once in an optimal solution (type 3 problems). Since each PCB will be loaded only once in an optimal solution, all components  $i \in \mathbb{N}_j$  will be loaded together.

#### 4. A heuristic model for the PCB-grouping problem

Since the general PCB-grouping problem is NP-complete, a heuristic solution algorithm is likely to be needed. This section outlines a simple heuristic that was used to find feasible solutions and was part of the branch and bound algorithm described in § 5. Branching in the branch and bound algorithm is based on whether two PCBs must be in the same group or must be in different groups. We note that a set of PCBs that, at some point of the branch and bound algorithm, must all be produced using the same group may be thought of as a mega-PCB whose component requirements are equal to the union of the sets of components ( $\mathbb{N}_j$ ) required by the PCBs that must be produced together (we refer to them simply as PCBs). If the cardinality of the union of these sets exceeds the capacity of the machine, there is no feasible solution satisfying the constraints that forced the PCBs to be produced together. We assume that the number of components required by any mega-PCB does not exceed the capacity of the machine. In fact, the branch and bound algorithm checks that this condition is satisfied before calling the heuristic algorithm.

The heuristic sequentially constructs groups of PCBs (and components), and is composed of the following steps (see Fig. 2).

- Step 1.* (Identify all mega-PCBs): The algorithm begins by identifying mega-PCBs based on the current set of constraints (if any) for pairs of PCBs that must be produced in the same group. After identifying the mega-PCBs, we will have a number of mega-PCBs and a number of PCBs which are not constrained to be produced with any other PCBs.
- Step 2.* (Begin a new group): Once all mega-PCBs are identified, a group is initiated with the unassigned PCB that requires the smallest number of components.
- Step 3.* (Identify and assign the unassigned PCBs): PCBs are added to the emerging group, beginning with the PCB with the smallest number of additional required components, until either (1) no unassigned PCBs remain, or (2) there are no more unassigned PCBs which can feasibly be assigned to the emerging group. Feasibility is based on: (1) the machine capacity,  $L$ , and (2) the set of constraints (if any) that preclude pairs of PCBs from being

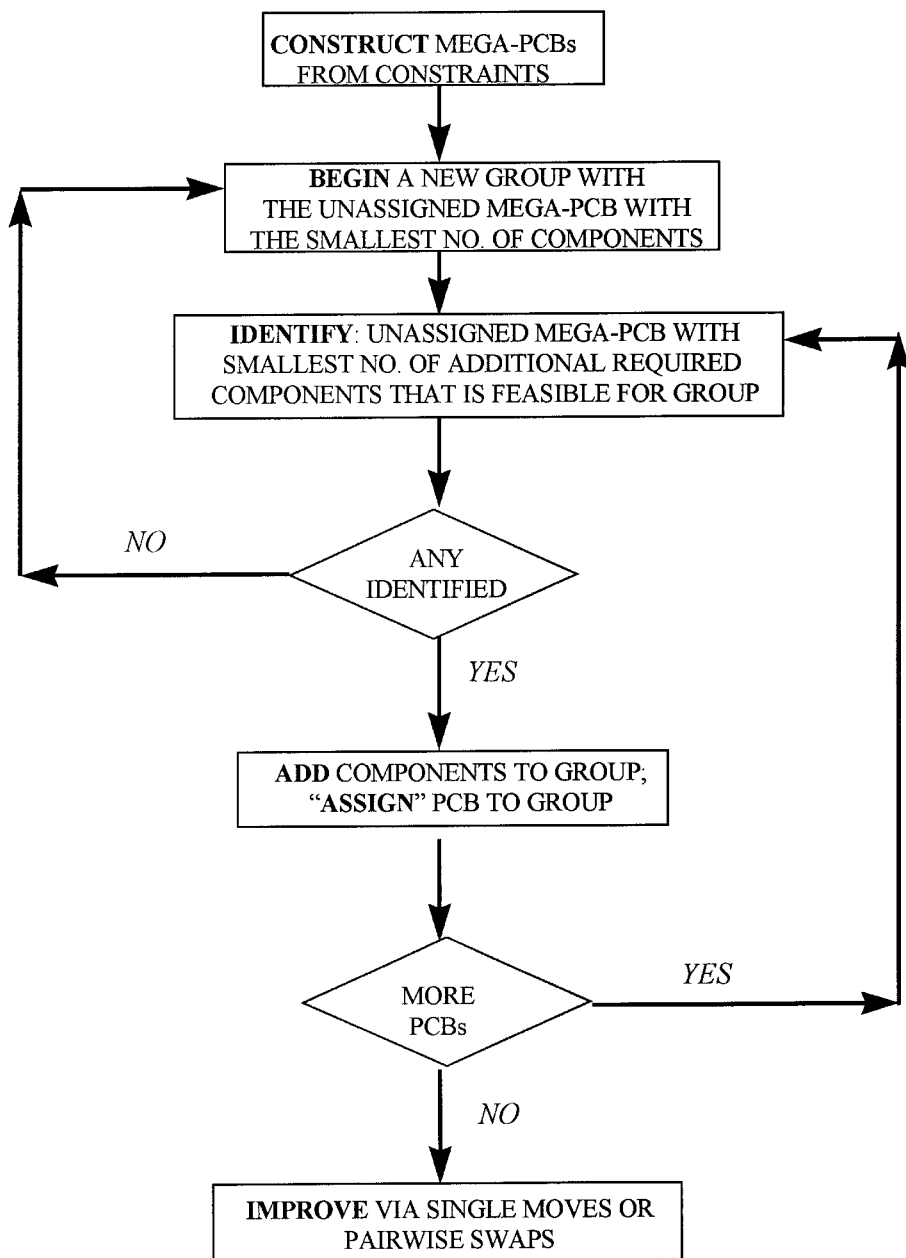


Figure 2. Flowchart of heuristic algorithm.

produced in the same group. If inclusion of a candidate PCB (or mega-PCB) would violate any of these constraints for a PCB already included in the emerging group, the candidate PCB is not included in the group. If inclusion of the candidate PCB in the emerging group would not have violated the machine capacity, the PCB with the next smallest number of additional required components is considered for inclusion in the emerging group (i.e., repeat *Step 3*). If no PCB may be added to the emerging group and

additional unassigned PCBs remain a new group is initiated (got to *Step 2*). If all PCBs or mega-PCBs have been assigned to a group, STOP.

After all PCBs are included in some group, we have a feasible solution to the problem. The heuristic algorithm includes the capability of performing single PCB moves and pairwise swaps of PCBs. A single PCB move consists of removing a PCB from its current group and evaluating the cost savings that would result from including the PCB in another group. If a feasible move of this sort can be identified and if the move results in a cost saving, the move is executed. Similarly, a pairwise swap consists of removing two PCBs from different groups and evaluating the cost savings that would result from including each PCB in the group in which the other was included. If a feasible swap of this sort can be identified and the swap reduces the total cost, the swap is executed. The heuristic algorithm alternates between attempting to perform a move and trying to perform a swap. Improvements continue until neither a feasible cost saving move nor a feasible cost saving swap can be identified.

## 5. A branch and bound algorithm

In developing a branch and bound algorithm the following issues need to be resolved:

- (1) The variable(s) on which to branch and the means of identifying the next variable on which to branch at each node of the tree;
- (2) the selection of the node in the tree from which to branch;
- (3) means of bounding the solution at each node; and,
- (4) when to apply upper bounding heuristics.

Each of these issues is briefly addressed in the subsections below. First, however, we summarize a domination property that allows us to identify pairs of PCBs which must be in the same group in an optimal solution.

### 5.1. A domination property

If the conditions of Theorem 1 hold and if  $N_j \subseteq N_k$ , then at least one optimal solution will have PCB  $j$  and PCB  $k$  in the same group. In this case we say that the PCB  $k$  dominates PCB  $j$ .

To prove this property, suppose we have an optimal solution in which PCB  $j$  and PCB  $k$  are in different groups, which we refer to as groups 1 and 2 respectively without loss of generality. Since each PCB is produced using exactly one group of components and since  $N_j \subseteq N_k$ , moving PCB  $j$  from group 1 to group 2 will not increase the number of components required to be loaded in group 2. Thus, the objective function cannot increase by such an exchange. If the objective function were to decrease as a result of such an exchange, the original solution could not have been optimal. Note that this domination property also holds if we replace  $N_k$  by  $\mathcal{N}_k$  (the set of component types  $i$  used by PCB  $k$  or any PCB forced to be in the same group as PCB  $k$ ).

This property allows us to force dominated/dominating pairs of PCBs together, thereby reducing the effective size of the problem by branching on these pairs of PCBs at the beginning of the branch and bound tree. It also allows us to fathom nodes of the tree for which this property is violated. Note that the optimal solution may involve producing PCBs  $k$  and  $h$  in different groups while  $N_j \subseteq \mathcal{N}_k$  and  $N_j \subseteq \mathcal{N}_h$ . The domination property implies only that at least one optimal solution

involves producing PCB  $j$  with either PCB  $k$  or PCB  $h$ . Thus, in using this property in the branch and bound algorithm, care must be taken not to fathom a node in which PCB  $j$  is forced not to be with PCB  $h$ , for example, if PCB  $j$  is already forced to be with PCB  $k$ .

### 5.2. Branching rules

Branching was done on whether or not a pair of PCBs must (or must not) be in the same group. In all cases, we branched from the rightmost unfathomed node in the tree.

To determine the next pair of PCBs on which to branch, we computed the following statistic for each pair of PCBs (provided PCB  $j$  is not dominated by PCB  $k$ ):

$$\begin{aligned}\phi_{jk} &= \text{the number of component types used by both PCB } j \text{ and PCB } k \\ &= \sum_i a_{ij} a_{ik}\end{aligned}$$

If PCB  $j$  is dominated by PCB  $k$ , we set  $\phi_{jk} = M$ , where  $M$  exceeds the number of components in the problem. Note that  $\phi_{jk}$  need only be computed once, before the beginning of the branch and bound procedure. The  $\phi_{jk}$  values were sorted into decreasing order. At any node in the branch and bound tree, the pair of PCBs on which the algorithm branched was the pair  $(j', k')$  with the largest  $\phi_{jk}$  value such that  $j'$  and  $k'$  were neither forced to be in the same group or forced to be in different groups by constraints that had been imposed further up the tree. Setting  $\phi_{jk} = M$  if PCB  $j$  is dominated by PCB  $k$  in essence forces this pair of PCBs to be in the same group since (1) the algorithm will branch on such pairs of PCBs before any other pairs are processed and (2) the branch in which they are constrained to be apart will be fathomed by the domination property outlined above.

### 5.3. Bounding the solution at each node

Since we restrict our attention to problems in which each PCB is loaded exactly once, a lower bound on the total cost can be obtained by multiplying a lower bound on the number of groups into which each component  $i$  must be included by the cost of loading component  $i$ ,  $s_i$ . At each node of the tree, four approaches were used to bound the number of groups in which component  $i$  must be included. The *first bound* is given by the total number of component types needed by all PCBs (or mega-PCBs) that require component type  $i$  divided by the maximum number of components that can be in a group,  $L$ , rounded up to the next larger integer (see appendix B.1). The *second bound* is given by the total number of PCBs ( $\in \mathcal{M}_i$ ) which, at some node in the branch and bound tree, are mutually forced to be in different groups (see appendix B.2). The *third bound* involves constructing a graph,  $G_i(V_i, E_i)$ , in which we create a vertex in the set  $V_i$  for every mega-PCB or unconstrained PCB (in the sense of not being forced to be in the same group as another PCB) that requires component type  $i$ . We create an edge between two vertices  $j$  and  $k$  in the graph if any PCB associated with vertex  $j$  is constrained to be in a different group from any PCB associated with vertex  $k$ . The chromatic number of this graph,  $\gamma(G_i)$ , (the minimum number of colours needed to colour each vertex such that no two vertices that are connected by an edge have the same colour) is a lower bound on the number of groups in which component type  $i$  must be included, based on the constraints at that

node of the branch and bound tree on pairs on PCBs that can and cannot be in the same group. Unfortunately, the chromatic number problem is itself an NP-complete problem. However, a number of lower bounds on the chromatic number have been proposed. All are functions of the number of vertices and edges in the graph. The best of these is the lower bound proposed by Ersov and Kozuhin (1962). See appendix B.3 for details. The *fourth bound* is based on the size of the maximum clique (a clique in graph  $G$  is a subgraph of  $G$  that is complete, i.e. each pair of vertices is connected by an edge) in a graph  $G_i$  whose construction was outlined above. If  $\rho(G_i)$  is the size of the maximum clique in graph  $G_i$ , then  $\gamma(G_i) \geq \rho(G_i)$ . Unfortunately again, the problem of finding the maximum clique in a graph is also NP-complete (Garey and Johnson 1979). However, in  $O(n^3)$  time we can check whether or not a clique of size 3 exists (in which case the fourth bound is set to 3; if not it is set to 0) simply by enumerating all possible combinations of 3 nodes and checking whether or not all three nodes are connected to each other. The actual lower bound on the number of times component type  $i$  must be loaded is the maximum of the four bounds outlined above.

#### 5.4. Fathoming nodes

Nodes in the tree can be fathomed in one of three ways. First, if the constraints that force pairs of PCBs to be in the same group create a mega-PCB whose component requirements exceed the capacity,  $L$ , of the machine, no feasible solution exists and the node may be fathomed. Second, if the component loading cost based on the lower bounds on the number of times each component must be loaded plus the sum of the PCB loading costs is equal to or greater than the value of a known solution, the node can be fathomed. Third, a left node (in which PCBs  $k$  and  $h$  are constrained to be in different groups) may be fathomed if (i) either  $k$  dominates  $h$  and  $h$  is not already dominated by some mega-PCB created further up the tree or (ii) vice versa. Similarly, a right node (in which PCBs  $k$  and  $h$  are constrained to be in the same group) may be fathomed if the newly created mega-PCB dominates some PCB  $j$  which is forced not to be in the same group as either  $k$  or  $h$  and PCB  $j$  is not already dominated by some mega-PCB created further up the tree.

#### 5.5. Application of the heuristic algorithm

The heuristic algorithm was applied (1) at the root node, (2) at nodes created by branching on a dominated/dominating PCB pair, and (3) when the list of  $\phi_{jk}$  values was exhausted. If the list of  $\phi_{jk}$  values is exhausted at some node in the branch and bound tree then there is no pair of PCBs that are not already either constrained to be in the same group or that are constrained to be in different groups. In that case, the heuristic algorithm simply evaluates the cost of the solution specified by the constraints that have already been imposed. That means that the heuristic solution at the node at which we are trying to branch may be treated as a lower bound on the cost at that node in the tree. Since the lower bound then equals or exceeds the value of the best known solution, the node may be fathomed.

### 6. Computational results using the branch and bound algorithm

In this section we outline results obtained from testing the branch and bound algorithm using four data sets. In Table 1 is a summary of key features of the four data sets. Data sets DATA1, CLUSTRA and CLUSTAX were drawn from industrial contexts. Data set DATA1 is listed in Maimon and Shtub (1991). DATA3 was

Data set name	DATA1	DATA3	CLUSTRA	CLUSTAX
Components				
Unique	43	43	15	59
Common	10	10	37	103
Total	53	53	52	162
PCBs				
Total number	8	16	26	34
Number undominated	8	16	17	22
PCBs/Common component				
Minimum	2	4	2	2
Average	3.50	7.00	4.73	5.92
Maximum	8	16	26	32
% of 1s in matrix				
Loading costs	18.4	13.3	14.1	12.1
All components	10	10	21	63
Tested range of machine capacities	1	1	1	1
	14—53	11—53	22—51	81—150

Table 1. Characteristics of input data sets

constructed from DATA1 in two steps. First, we created a duplicate copy of each PCB, thereby generating a problem with 16 PCBs. In the resulting matrix each component would be a shared component. Therefore, we removed one of the occurrences of each of the unique components in DATA1 from the matrix generated by creating a copy of each PCB. In removing these components, we alternated between removing the component from the original PCB (number 1–8) and the generated PCB (numbered 9–16). In this way, we created a problem which again had 43 unique components, and 10 shared components. For each row corresponding to a shared PCB, the density of ones was the same in DATA1 and DATA3.

The size of the problems analysed ranged from 8 PCBs and 53 components (with only 10 shared components) for DATA1 to 34 PCBs and 162 components (with 103 shared components) for CLUSTAX. The density of ones in the matrices ranged from 12.1% (CLUSTAX) to 18.4% (DATA1).

### 6.1. Results for DATA1 and DATA3

Table 2 is a summary of the key results for DATA1 and DATA3. We focus our attention on the data set DATA3 because it includes problems with larger size of the PCB-component input matrix, which were the most difficult problems to solve using our approach. The results for DATA3 are broken into two groups based on the machine capacity. Results for capacities 11–13 are reported separately from those for capacities 14–53 for two reasons. First, the solution times for the small capacity problems were all very large. Second, by separating the results for capacities in the range 14–53, we can more readily compare the results with those found for DATA1. The results for capacities 11–13 as well as the entire data set are shown in Table 2. The entire data set (capacity range 11–53) illustrates the effect of each capacity range. Larger capacity problems shift the results (e.g. solution time) towards significantly improved performance.

For DATA1, all solution times were under 1.05 seconds and the average was under 0.3 s; for DATA3 (with capacity range  $14 \leq L \leq 53$ ), the maximum solution time was just under 1.76 minutes and the average was under 8.34 s. Table 2 also

Data set name	DATA1	DATA3	DATA3	DATA3
Capacity range	14—53	11—13	14—53	11—53
Solution time*				
Minimum	0:10	462:58	0:22	0:11
Average	0:30	639:72	8:34	52:39
Maximum	1:05	986:90	105:24	986:90
Nodes evaluated				
Minimum	1	25 223	1	1
Average	36	36 810	514	3 047
Maximum	153	58 909	6069	58 909
Total enumeration	2·86E + 08	1·33E + 36	1·33E + 36	1·33E + 36
% Runs in which initial heuristic was optimal				
w/Move and swap	92·5%	0·0%	77·5%	72·1%
w/o Move and swap	55·0%	0·0%	5·0%	4·7%

\*Seconds on Zenith 386/16 computer with an 80387 math coprocessor. Code was written in Turbo PASCAL version 5.5.

Table 2. Summary of results for DATA1 and DATA3.

compares the number of nodes examined in the branch and bound tree to the number of solutions that would have had to be examined had total enumeration been employed. Even in the worst case, only a minuscule fraction of the total number of solutions needed to be examined.

Finally, in over 77·5% of the runs, the initial heuristic solution was optimal when the move and swap improvement procedures were used. When they were not employed, this percentage dropped significantly, particularly for DATA3.

Figure 3 is a plot of the percentage of the total number of examined nodes that had to be explored before the optimal solution was found versus the machine capacity,  $L$ . Most (about 90%) of the computational effort is devoted to proving that a solution obtained early in the branch and bound process is, in fact, optimal. In the few cases in which the percentage shown in Fig. 3 is large, the number of nodes examined is very small.

Figure 4 is a plot of the percentage improvement in the solution that results from (1) the use of the move and swap algorithms and (2) the use of the branch and bound algorithm as a function of the machine capacity,  $L$ , for DATA3 (with PCB-component matrix as defined above). Improvement percentages are measured relative to the optimal value. Most of the improvement of the initial heuristic solution was due to the move and swap procedures and not to the branch and bound algorithm. In many cases, particularly for larger machine capacities, the branch and bound algorithm was only used to confirm the optimality of the solution obtained from the initial heuristic followed by the move and swap procedures. Similar results were obtained for DATA1.

## 6.2. Results for CLUSTRA and CLUSTAX

For CLUSTRA and CLUSTAX, in addition to solving the problems to optimality, we solved the problems by fathoming nodes in the tree whenever the differences between the best known upper bound and the lower bound was less than  $\varepsilon$  times the

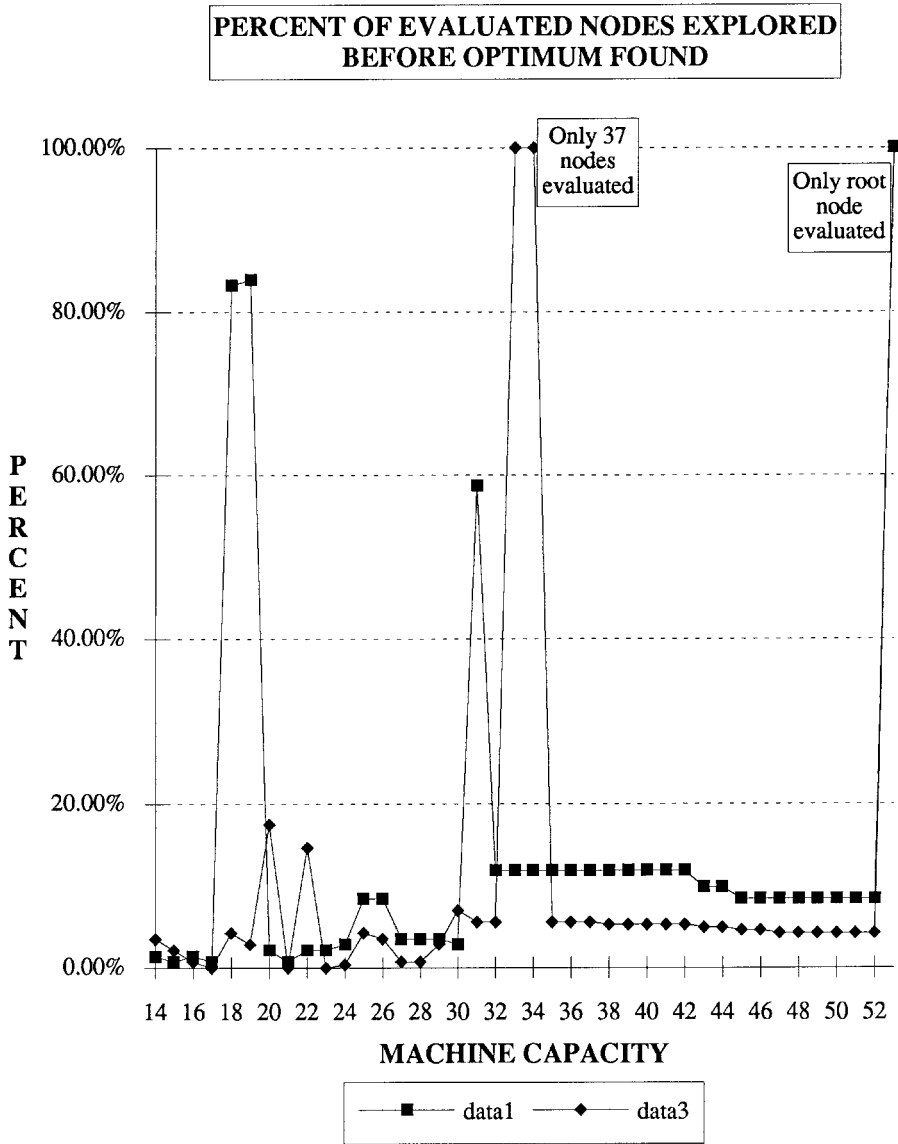


Figure 3. Percentage of evaluated nodes explored before optimum found.

lower bound. Two values of  $\epsilon$  were used (0.02 and 0.01) as shown in Table 3. For each data set, the PCB loading cost (see Table 1) was set to the smallest value satisfying the conditions of Theorem 1. Thus, the constant PCB loading cost was as small as possible. Larger PCB loading costs would have made the  $\epsilon$ -optimal problems easier to solve since the constant PCB cost was included in the lower and upper bounds on the objective function. Finally, instead of running each problem as an independent run, the best solution found for the problem with a machine capacity of  $L$  was taken as a feasible solution at the root node of the branch and bound tree for the problem with a capacity of  $L + 1$ . This solution was compared to the heuristic solution (for a capacity of  $L + 1$ ) and the smaller value was used as the

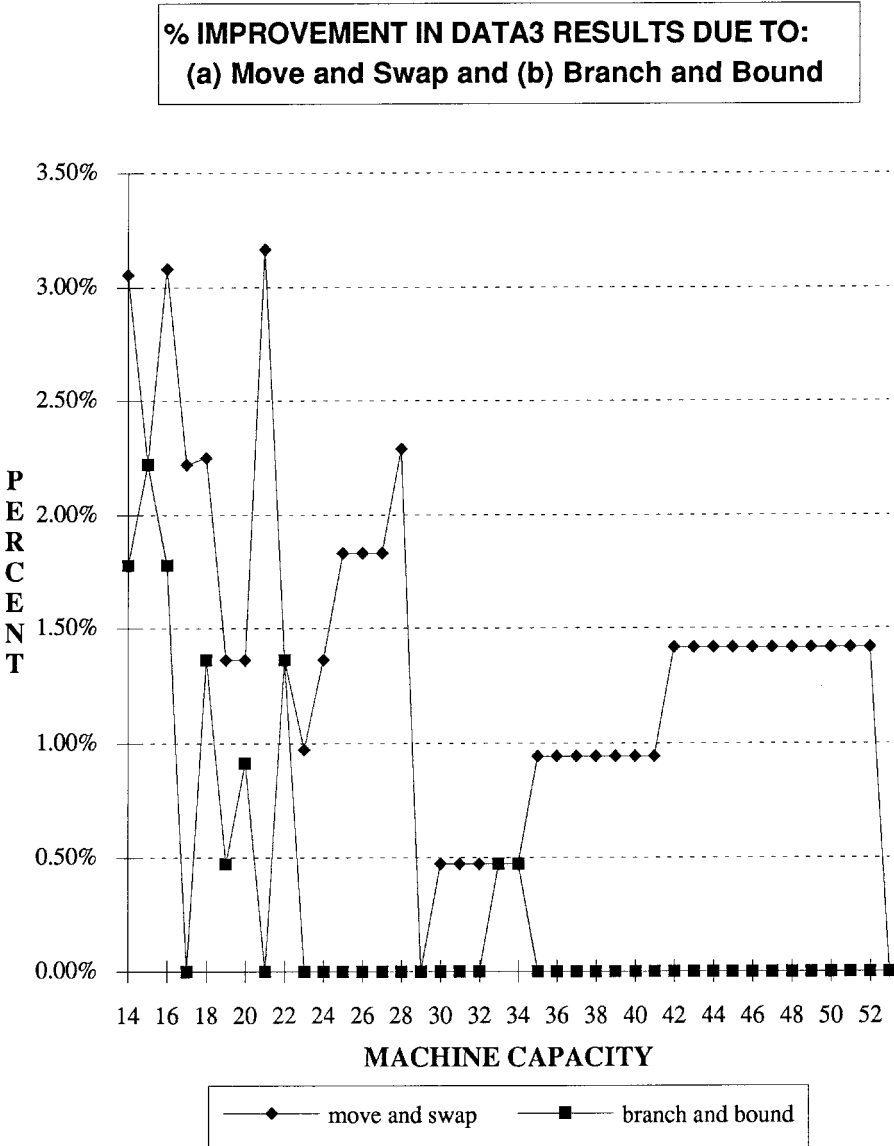


Figure 4. Percentage improvement in Data3 results due to (a) move and swap and (b) branch and bound.

upper bound at the root node of the branch and bound tree for the problem with a capacity of  $L + 1$ . Thus, successive runs for these data sets are not independent.

For both data sets, both the solution time and the number of nodes examined in the branch and bound tree increase dramatically as  $\epsilon$  decreases. The time required to solve the problems optimally averaged under 2 minutes for CLUSTRA and just over 40 minutes for CLUSTAX. For  $\epsilon = 0.01$ , these times dropped to 18 and 75 seconds respectively. As expected, the percentage of runs in which the branch and bound algorithm was needed (either to improve on the root node solution or to confirm its

Data set name	CLUSTRA	CLUSTRA	CLUSTRA	CLUSTRA	CLUSTAX	CLUSTAX	CLUSTAX
$\epsilon$ -Optimality	0	0-01	0-02	0	0-01	0-02	0-02
Solution time*							
Minimum	0-22	0-33	0-27	50-10	8-84	1-15	1-15
Average	96-05	17-19	3-12	2 407-91	75-38	2-55	2-55
Maximum	1239-28	336-37	33-45	26 324-52	1192-93	12-52	12-52
Nodes evaluated							
Minimum	1	1	1	259	27	1	1
Average	2 410	417	47	21 279	616	7	7
Maximum	34 857	8905	783	243 353	10 885	85	85
Total Enumeration	6-84E + 97	6-84E + 97	6-84E + 97	7-55E + 168	7-55E + 168	7-55E + 168	7-55E + 168
% R runs in which branch and bound algorithm needed	96-8%	83-9%	67-7%	100-0%	100-0%	10-0%	10-0%
% Component cost over minimum							
Minimum		0-00%	0-00%		0-00%	0-00%	0-00%
Average		2-62%	4-95%		3-04%	3-04%	3-04%
Maximum		8-20%	20-00%		11-23%	11-23%	11-23%

\*Seconds on Zenith 386/16 Computer with an 80387 math coprocessor.  
Code was written in Turbo PASCAL version 5.5.

Table 3. Summary of results for CLUSTRA and CLUSTAX.

$\varepsilon$ -optimality) decreased as  $\varepsilon$  increased. Finally, Table 3 reports the actual % deviation from optimality of the component loading cost term in the objective function (the term over which optimization was possible). The largest percentage deviations tended to occur at the largest machine capacities where the component loading cost term was the smallest. However, these problems also tended to be the easiest to solve to optimality. Thus, in practice it may be feasible simply to solve the large capacity problems optimally or with very small values of  $\varepsilon$  to reduce the deviation of the component loading cost term from its optimal value.

## 7. Directions for future work

The model described in this paper can be extended in a number of ways. First, algorithms designed to solve the problem of simultaneously determining a set of permanent components and a number of different component groups should be developed. Second, most industrial concerns employ multiple machines in the fabrication of PCBs. Each machine may have its own unique production rate and capacity. Thus, in addition to the problem of grouping PCBs and components, analysts must assign PCBs and component groups to machines. Third, the model analysed in this paper deals only with grouping components. It does not deal with the number of times each component is required by each PCB nor does it deal with the time required to assemble a PCB. Incorporation of these factors along with cycle inventory considerations would enhance the usefulness of the model. Finally, underlying assumptions of the model are that the component loading cost is incurred whenever a new group of components is loaded on the machine. Based on at least one plant with which the authors are familiar, this is not a bad assumption. A group of components is completely removed from the machine before a new group is loaded. However, if some components are common to the group being removed from the machine and the group being put on the machine, the component loading cost may be reduced by leaving the common components on the machine. These issues related to the sequencing of groups and further complicate the problem.

## Acknowledgments

Daskin's work was supported by a Fulbright Grant. The financial support is gratefully acknowledged. The authors wish to thank two anonymous referees for their useful comments and help in improving the paper.

## Appendix A. Proof of Theorem 2

First, the PCB-grouping problem is in NP since, given any solution (defined by values of the  $O(I^2 \cdot J^2)$  decision variables), we can check in polynomial time whether or not the constraints are satisfied and whether or not the objective function is less than or equal to some value  $Z$ . Note that there will be at most  $I \cdot J$  groups, where  $I$  is the number of components and  $J$  is the number of PCBs. This is so because each PCB will be loaded at most  $I$  times (for problems that do not satisfy the conditions of Theorem 1) and each component will be loaded at most  $J$  times. This bound will be realized if each PCB uses only unique components (i.e.,  $O_j = N_j, \forall j$ ) and  $L = 1$ . (Note that if the conditions of Theorem 1 are satisfied, there will be at most  $J$  groups and the number of decision variables will be  $O(I \cdot J^2)$ ). Finally, we note that in practical problems the number of groups will usually be smaller than either  $I$  or  $J$ .)

Next, we show that there is a polynomial transformation from the 3-Partition problem to an instance of the PCB-grouping problem, such that there is a solution to

the 3-Partition problem if and only if there is a solution to the PCB-grouping problem.

For  $j = 1, \dots, 3m$  (note that  $J = 3m$  in this instance of the PCB-grouping problem) let:

$$D_0 = 0 \text{ and } D_j = \sum_{m=1}^j d_m$$

Define the following sets of indices of component types  $i$  for each of the  $3m$  PCBs:

$$C_j = \{mB + 1\}; \quad O_j = \{1 + D_{j-1}, \dots, D_j\}; \\ N_j = C_j \cup O_j = \{1 + d_{j-1}, \dots, D_j\} \cup \{mB + 1\}$$

Finally, we set:  $S_j = 2\forall j$ ;  $s_i = 1\forall i$ ;  $L = B + 1$  and  $Z = \sum_i d_i + 7m$ .

Since the 3-Partition problem is NP-complete for values of  $B$  that are bounded by a polynomial function of  $m$ , this transformation can be accomplished in polynomial time. We have created an instance of the PCB-grouping problem in which each of the first  $mB$  components is used in only one PCB and the last component, component  $mB + 1$ , is used in every PCB. Thus,  $|C_j| = 1\forall j$  and  $|N_j| = d_j + 1 < B + 1 = L\forall j$  and therefore, by Theorem 1, each PCB will be loaded exactly once in an optimal solution since  $2 = S_j > \sum_{i \in C_j} s_i = 1\forall j$ .

Finally, we show that if we have a solution to this instance of the PCB-grouping problem with an objective function value less than or equal to  $Z = \sum_i d_i + 7m$ , then we can construct a solution to the corresponding instance of the 3-Partition problem. First, we show that in an optimal solution to the PCB-grouping problem with this objective function, each group must have exactly  $L = B + 1$  components. Suppose this is not so. Then there must exist some group with  $B$  or fewer components. Without loss of generality let one such group be group 1. Then, since component  $mB + 1$  must be in every group, the number of the first  $mB$  components in group 1 must be less than or equal to  $B - 1$ . Therefore, the number of the first  $mB$  components that must be in the remaining groups must be greater than or equal to  $mB - (B - 1) = (m - 1)B + 1$ . Again, since component  $mB + 1$  must be in every group, there is only room for  $B$  other components in each group. Thus, the remaining  $(m - 1)B + 1$  components can not be in only  $m - 1$  groups. That is, they must be in at least  $m$  additional groups and the number of groups must therefore be at least  $m + 1$ . However, if this is so, each of the  $mB$  unique components will be loaded exactly once and the common component (component  $mB + 1$ ) must be loaded at least  $m + 1$  times. Thus, the component loading cost will be  $\sum_i d_i + m + 1$ . Each PCB  $j$  will be producible by one group  $g$  (the group  $g$  such that  $j \in P_g$ ). Thus, each PCB will be loaded only once and the PCB loading cost will be  $\sum_{j=1}^{3m} S_j = 6m$ . Consequently, the total cost of such a solution must be at least  $\sum_i d_i + (m + 1) + 6m = \sum_i d_i + 7m + 1 > Z$ . Therefore, each group must contain exactly  $L = B + 1$  components.

Because of the relative values of  $S_j$  and  $s_i$  and the fact that  $|N_j| \leq L\forall j$ , Theorem 1 states that each PCB  $j$  will be loaded exactly once (in a single group  $g$ ) in an optimal solution. Therefore, for the  $(j, g)$  pair, we have  $X_{jg} = 1\forall i \in N_j$ . Also,  $Z_{ig} = 1$  if  $\exists j$  such that PCB  $j$  is produced in group  $g$  (i.e.,  $Y_{jg} = 1$ ) and  $i \in N_j$ , and  $Z_{ig} = 0$  otherwise. By construction, the set  $N_j$  consists of  $d_j$  unique components (numbered  $1 + D_{j-1}$  through  $D_j$ ) plus component  $mB + 1$ . Thus, if PCB  $j$  is produced using group  $g$ , we include index  $j$  in  $P_g$ . Since each PCB  $j$  will be in only one group

and since the total number of unique components in each group is  $B$ , this assignment of indices  $j$  to sets  $P_g$  will result in  $\sum_{j \in P_g} d_j = B$  for all sets  $P_g$ . Hence, the assignment will be a solution to the 3-Partition problem.

**Appendix B. Bounds on the number of times component type  $i$  must be loaded**

**B.1. First lower bound:**

Let us introduce the following notation:

$$\delta(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{otherwise} \end{cases}$$

$\mathcal{M}_i$  = the set of indices of PCBs  $j$  that use component type  $i$  or that are part of mega-PCB any one of whose constituent PCBs uses component type  $i$

With this notation, a lower bound on the total number of component type  $i$  usage is given by:

$$U_i = \sum_k \delta\left(\sum_{j \in \mathcal{M}_i} a_{kj}\right) \tag{11}$$

The inner summation equals the number of PCBs that use both component type  $k$  and component type  $i$ . The  $\delta(\cdot)$  function sets this total to 1 if there is at least one such PCB. Summing over all component types  $k$ ,  $U_i$  is the total number of component types needed by all PCBs that require component type  $i$ . If this number is less than or equal to the machine capacity,  $L$ , then all of these PCBs could, at least from the perspective of component type  $i$  be in the same group. In the case,  $U_i$  indicates that component type  $i$  need only be loaded once. More generally,

$$\left\lceil \frac{U_i}{L} \right\rceil \tag{12}$$

is a lower bound on the number of times component type  $i$  must be loaded, where  $\lceil x \rceil$  is the smallest integer greater than or equal to  $x$ .

**B.2. Second lower bound**

Let  $j_1$  and  $j_2$  be two PCBs such that  $j_1, j_2 \in \mathcal{M}_i$ . If, at some node in the branch and bound tree, PCBs  $j_1$  and  $J_2$  are forced to be in different groups, then component type  $i$  must be loaded at least twice. We note that  $j_1$  and  $j_2$  may be forced to be in different groups implicitly. For example,  $j_1$  may be forced to be in the same group as some PCB  $j_3$ ,  $j_2$  may be forced to be in the same group as PCB  $j_4$  and, PCBs  $j_3$  and  $j_4$  may be constrained to be in different groups. This would implicitly constrain  $j_1$  and  $j_2$  to be in different groups.

**B.3. Third lower bound**

Ersov and Kozuin (1962) showed that

$$\left\lceil \frac{n}{\lfloor t \rfloor} \left( 1 - \frac{t - \lfloor t \rfloor}{1 + \lfloor t \rfloor} \right) \right\rceil \tag{13}$$

is a lower bound on the chromatic number of a graph, where  $n = |V_i|$ ,  $p = |E_i|$ ,  $t = n - 2p/n$  and  $\lfloor x \rfloor$  is the largest integer  $\leq x$ . Ersov and Kozuhin (1962) outlined the construction of graphs with  $n$  vertices and  $p$  edges that attain this lower bound.

However, it should be noted that (13) is computationally intensive. This is important because the bound may need to be computed for each common component type  $i$  at each node of the branch and bound tree. In fact, the bound is likely to be computed multiple times for each common component type at each node in the tree.

To alleviate the computational cost which is involved in computing (13), we compute chromatic numbers of graphs  $G'$  that are constructed by deleting vertices and any edges incident on the deleted vertices from  $G$ . This relationship is worth noting because the lower bounds on  $G'$  using (13) may actually be *tighter* than this obtained for  $G$ . This is so because  $\gamma(G)$  is also an upper bound on  $\gamma(G')$ . Thus, a lower bound on the chromatic number of  $G'$  is also a lower bound on the chromatic number of  $G$ .

To summarize, the process of successfully deleting the vertex (and incident edges) with the smallest degree coupled with the bound provided by (13) constitutes the third lower bound. The process is continued until a graph is constructed in which its number of vertices  $n$  exceeds the best lower bound which has hitherto been computed. This process of vertices  $n$  exceeds the best lower bound which has hitherto been computed. This process proved to be very effective for problems with small machine capacities, which were the most difficult problems to solve using our approach.

## References

- ASKIN, R. G. and CHIU, K. S., 1990, A graph partitioning procedure for machine assignment and cell formation in group technology. *International Journal of Production Research*, **28**, 1555–1572.
- BOOTHROYD, G., 1992, *Assembly Automation and Product Design* (New York: Marcel Dekker).
- CHANDRASEKHARAN, M. P. and RAJAGOPALAN, R., 1986a, An ideal seed non-hierarchical clustering algorithm for cellular manufacturing. *International Journal of Production Research*, **24**, 451–464.
- CHANDRASEKHARAN, M. P. and RAJAGOPALAN, R., 1986b, MODROC: An extension of rank order clustering for group technology. *International Journal of Production Research*, **24**, 1221–1233.
- CHU, C. H., 1993, Manufacturing cell formation by competitive learning. *European Journal of Operational Research*, **69** (3), 292–311.
- CHUNG, Y. and KUSIAK, A., 1994, Grouping parts with a neural network. *Journal of Manufacturing Systems*, **13**, 262–275.
- ERSOV, A. P. and KOZUHIN, G. I., 1962, Estimates of the chromatic number of connected graphs. *Soviet Mathematics*, Translation of Doklady Akademii Nauk SSSR, **3**, 50–53.
- GAREY, M. R. and JOHNSON, D. S., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (San Francisco, CA: Freeman).
- HARHALAKIS, G., NAGI, R. and PROTH, J. M., 1990, An efficient heuristic in manufacturing cell formation for group technology applications. *International Journal of Production Research*, **28**, 185–198.
- HARHALAKIS, G., PROTH, J. M. and XIE, X. L., 1990, Manufacturing cell design using simulated annealing: an industrial application. *Journal of Intelligent Manufacturing*, **1**, 185–191.
- KAPARTHI, S., SURESH, N. C. and CERVENY, R. P., 1993, An improved neural network leader algorithm for part-machine group technology. *European Journal of Operational Research*, **69**, 342–356.
- KING, J. R., 1980a, Machine-component group formation in group technology. *OMEGA: The International Journal of Management Science*, **8**, 193–199.
- KING, J. R., 1980b, Machine-component grouping in production flow analysis: an approach using a rank-order clustering algorithm. *International Journal of Production Research*, **18**, 213–232.

- KUMAR, K. R., KUSIAK, A. and VANNELLI, A., 1986, Grouping of parts and components in flexible manufacturing systems. *European Journal of Operational Research*, **24**, 387–397.
- KUSIAK, A., 1987, The generalized group technology concept. *International Journal of Production Research*, **25**, 561–569.
- LIU, C. M. and WU, J. K., 1993, Machine cell formation: using the simulated annealing algorithm. *International Journal of Computer Integrated Manufacturing*, **6** (6), 335–349.
- MAIMON, O. and SHTUB, A., 1991, Grouping methods for printed circuit boards assembly. *International Journal of Production Research*, **29**, 1379–1390.
- MCGINNIS, L. F., AMMONS, J. C., CARLYLE, M., RANMER, L., DEPUY, G. W., ELLIS, K. P., TOVEY, C. A. and XU, H., 1992, Automated process planning for printed circuit card assembly. *IIE Transactions*, **24**, 18–26.
- PAPADIMITRIOU, C. H. and STEIGLITZ, K., 1982, *Combinatorial Optimization: Algorithms and Complexity* (Englewood Cliffs, NJ: Prentice Hall).
- PIERREVAL, H. and PLAQUIN, M. F., 1994, A genetic algorithm approach to group machines into manufacturing cells. *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pp. 267–271.
- RAJAGOPALAN, R. and BATRA, J. L., 1975, Design of cellular production systems: a graph theoretic approach. *International Journal of Production Research*, **13**, 567–579.
- ROCKWELL, T. H. and WILHELM, W. E., 1990, Material flow management in cellular configurations for small-lot circuit card assembly. *International Journal of Production Research*, **28**, 573–594.
- SHAFFER, S. M. and MEREDITH, J. R., 1990, A comparison of selected manufacturing cell formation techniques. *International Journal of Production Research*, **28**, 661–673.
- SHTUB, A., 1989, Modeling group technology cell formation as a generalized assignment problem. *International Journal of Production Research*, **27**, 775–782.
- SINGH, N., 1993, Design of cellular manufacturing systems: an invited review. *European Journal of Operational Research*, **69**, 284–291.
- VANNELLI, A. and KUMAR, K. R., 1986, A method for finding minimal bottle-neck cells for group part-machine families. *International Journal of Production Research*, **24**, 387–400.