INT. J. PROD. RES., 1 DECEMBER 2004,
VOL. 42, NO. 23, 5031–5048

Taylor & Francis
Taylor & Francis Group

Optimal algorithm for minimizing production cycle time of a printed circuit board assembly line

D. M. KODEK* and M. KRISPER

The problem of the optimal allocation of components to a printed circuit board assembly line will several non-identical placement machines in series is considered. The objective is to achieve the highest throughput by minimizing the production cycle time of the assembly line. This problem can be formulated as a minimax approximation integer programming problem that belongs to the family of scheduling problems. The difficulty lies in the fact that this problem is proven to be NP-complete. All known algorithms are exponential and work only if the number of variables is reasonably small. This particular problem, however, has properties that allow the development of a very efficient type of branch-and-bound-based optimal algorithm that works for problems with a practically useful number of variables. Detailed description of the algorithm is given together with examples that demonstrate its effectiveness.

1. Introduction

In a printed circuit board (PCB) assembly line, the boards usually travel on a conveyer belt through a line of component placement machines, each placing a number of components on the circuit board. Assuming this type of production set-up, the planning decisions can be divided into the following two subproblems (Ammons *et al*. 1997):

(1) Component allocation: decide which component types are placed by which machine.
(2) Feeder arrangement and placement sequencing: stage component feeders on each machine and sequence the placement operations for each machine and card type.

These subproblems are not completely independent. The placement time provided by a machine's manufacturer is usually the fastest time with which a component type can be placed. These times are not always realized because of latency, which is determined by the feeder arrangement and placement sequencing. Of course, feeder arrangement and placement sequencing cannot be decided without component allocation. Clearly, there is a circular interaction between the subproblems. The typical strategy (Askin *et al*. 1994, DePuy et al. 1997, Schtub and Maimon 1992) is to first solve the component allocation problem and then solve the feeder arrangement/placement sequencing problem.

Revision received June 2004.

Faculty of Computer Information Science, University of Ljubljana, Traška 25, 1000 Ljubljana, Slovenia.
*To whom correspondence should be addressed. E-mail: duke@fri.uni-lj.si

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540412331285814

This paper only addresses the problem of optimal component allocation. This problem is NP-complete and is often considered too difficult to solve optimally in practice. This opinion is supported by the experience with the general integer programming software that is typically very slow and does not produce solutions in a reasonable time. It is therefore not surprising to see attempts of replacing the optimal solution with a near-optimal one (Ji *et al.* 2001, Kim *et al.* 1996). The reasoning is as follows. A near-optimal solution is often good enough and is usually obtained in a significantly shorter time than the optimal solution. Although this is true in many cases, it does not hold always. One difficulty with the near-optimal methods is that, as a rule, they do not give an estimate of closeness to the optimal solution. This means that a significantly better optimal solution, about which the user knows nothing, may exist.

The paper presents a new optimal algorithm that takes advantage of the special properties of the minimax approximation allocation problem. This algorithm is much faster than the general integer programming approach mentioned above. It finds, in many practical cases, the optimal solution in a time similar to the time needed for near-optimal methods. Because of NP-completeness, it will not find the optimal solution for the cases with a large number of variables, but it always produces near-optimal solutions that can be used in such cases.

2. Formulation of the problem

There are two important differences between the traditional assembly line problem and the PCB component allocation problem. First, unlike the traditional assembly line, the precedence of operations in the PCB assembly is generally not important and will be ignored. The second difference concerns the assembly times for the same component on different machines. Due to various types and configurations of the placement machines, different machines have different times for placement of the same kind of component.

An example from table 1 describes a PCB assembly line with three different placement machines M_1, M_2, M_3 and a board with 10 types of components. The placement times t_{ij} for different components and machines are also given. If a machine cannot handle a particular type of component, its placement time is assigned to be infinite (∞). The infinity is used here for simplicity of notation only — it is replaced by a large positive number for computation. In addition to the time needed to place a component, there is also a set-up time s_i for each of the

	Placement times t_{ij} for component type j										
Machine M_i	1	2	3	4	5	6	7	8	9	10	Set-up time s_i
1	0.3	0.7	0.7	0.5	∞	∞	∞	0.6	0.5	∞	5.0
2	0.7	1.2	1.5	1.6	1.5	1.8	1.9	∞	0.7	1.9	6.7
3	1.7	2.3	2.2	2.4	1.9	1.5	2.0	2.5	∞	1.8	7.8
Number c_j of components	162	90	51	30	22	16	12	9	7	5	

Table 1. Example of a PCB assembly line with three different placement machines and 10 different component types per board. The placement times t_{ij} for different component and machines and the setup times s_i are in seconds.

machines M_i. Finally, a total number of each type of a component per board c_j is given.

Obviously, there are many possible ways of allocating the components j to the placement machines M_i. The question is how to allocate the components in such a way that the assembly line has the best performance. The PCB assembly line cycle time T is formally defined as the maximum time needed by one of the machines M_i, $i = 1, 2, \ldots, m$, to complete the placement of the components allocated to it.

Suppose that there are m non-identical placement machines M_i in a PCB assembly line and that a board with n types of components is to be assembled on this line. The component allocation problem can be formally defined as

$$T_{\text{opt}} = \min_{x_{ij}} \max_{i=1,2,\ldots,m} \left(s_i + \sum_{j=1}^{n} t_{ij} x_{ij} \right), \tag{1}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = c_j, \quad j = 1, 2, \ldots, n, \tag{2}$$

$$x_{ij} \geq 0 \quad \text{and integer.} \tag{3}$$

The solution is the optimal cycle time T_{opt} and the optimal allocation variables $x_{ij}^{(\text{opt})}$. The variable x_{ij} gives the number of components of type j allocated to machine M_i. Constraints (2) ensure that all of the components will be allocated. The components are indivisible and (3) ensures that x_{ij} are positive integers. Note that we ignore the feeder capacity limits and other machine constraints.

It is the integer constraint on x_{ij} that makes the problem (1–3) difficult. Solving it without this constraint is quite simple, even for a very large number of variables x_{ij}. Rounding the non-integer x_{ij} to the nearest integers is certainly a plausible strategy and is used in practice. There are, however, limitations to this approach. A significantly better solution may exist and it is not known how close to the optimal solution the rounded one is.

3. Complexity of the problem

Problem (1–3) is a combination of assignment and flowshop scheduling problems (Brucker 1998) and is NP-complete for $n \geq 2$. Proving the NP-completeness is not too difficult. First, it is trivial to show that the problem is in P. Second, it is possible to show that the well known PARTITION problem can be polynomially transformed into (1–3) (Vilfan 2002). Since PARTITION is NP-complete, so is our problem.

A typical approach to solving this problem is to treat it as a general mixed-integer linear programming problem. The minimax problem (1–3) is reformulated as

$$\begin{gathered} \min_{x_{ij}} T_{\text{opt}}, \\ T_{\text{opt}} - s_i - \sum_{j=1}^{n} t_{ij} x_{ij} \geq 0, \quad i = 1, 2, \ldots, m, \\ \sum_{i=1}^{m} x_{ij} = c_j, \quad j = 1, 2, \ldots, n, \\ x_{ij} \geq 0 \quad \text{and integer.} \end{gathered} \tag{4}$$

All algorithms capable of solving this problem optimally work by starting with the non-integer problem where the variables x_{ij} can be any positive real number. Additional constraints are then gradually introduced into the problem and these constraints eventually force the variables x_{ij} to integer values. Many instances of suitably reformulated subproblems of the form (4) must be solved before the optimal solution is found.

An advantage of formulation (4) is that general mixed-integer programming software can be used to solve it. Unfortunately, this advantage occurs at the expense of computation time. The general software uses the simplex algorithm to solve the subproblems. The simplex algorithm is very general and slow since it does not use any of the special properties of the minimax problem. All these properties are lost if the original problem (1–3) is converted into the general problem.

It is the purpose of this paper to develop an optimal algorithm that does not use the generalized formulation (4). The algorithm takes advantage of the special properties of the minimax problem (1–3) and avoids using the simplex algorithm completely.

4. Lower bound theorem

The basic idea of our algorithm is to use a lower bound for T_{opt} as a tool that leads to the solution. This lower bound must be computed for each of the subproblems that appear within the branch-and-bound process. It must take into account the fact that some of the subproblem's variables x_{ij} are known integers. To derive it, assume that the subproblems's variables $x_{ij}, j = 1, 2, \ldots, k-1$, are known integers for all i. In addition, some, but not all, of the variables x_{ik} may also be known integers. Let I_k be the set of indices i that correspond to the known integers x_{ik}. The subproblem's variables can be formally described as

$$x_{ij} = \begin{cases} x_{ij}^I, & j = 1, \ldots, k-1, \quad i = 1, \ldots, m \\ x_{ij}^I, & j = k, \quad i \in I_k \\ x_{ij}, & j = k, \quad i \notin I_k \\ x_{ij}, & j = k+1, \ldots, n, \quad i = 1, \ldots, m, \end{cases} \tag{5}$$

where k can be any of the indices $1, 2, \ldots, n$. Notation x_{ij}^I is used to describe the variables that are already known integers. The remaining variables x_{ij} are not yet known. The number of indices in the set I_k lies in the range 0 to $m-2$. If there were $m-1$ known integers x_{ik}^I the constraint (2) gives the remaining variable which contradicts the assumption that not all of the variables x_{ik} are known. The index k changes to $k+1$ when all x_{ik} are known integers.

Definition (5) assumes that a certain rule is used to introduce the constraints, which force the variables x_{ij} to integer values. This rule is simple: for every index k, it is necessary to constrain x_{ik} to known integers x_{ik}^I for all $i, i = 1, 2, \ldots, m$, before k can change. The rule follows from the structure of constraints given by (2) and is needed to derive the lower-bound theorem. There is no problem with this rule because the branch-and-bound method on which our algorithm is based allows complete freedom of choosing the variable x_{ij} that is next to be constrained. The indices k can be selected in any order. A simple ascending order $k = 1, 2, \ldots, n$, is used in (5). This also applies to the case when the problem is first reordered along the indices j in a way that gives the fastest rate of lower bound increase. Such a reordering is used in our algorithm.

To simplify the notation, let us first use the known integers x_{ij}^I and redefine s_i as s_i'

$$s_i' = \begin{cases} s_i + \sum_{j=1}^{k} t_{ij} x_{ij}^I, & i \in I_k \\ s_i + \sum_{j=1}^{k-1} t_{ij} x_{ij}^I, & i \notin I_k. \end{cases} \tag{6}$$

Similarly, the known integers x_{ik}^I (if any) are used to redefine c_k as c_k'

$$c_k' = c_k - \sum_{i \in I_k} x_{ik}^I. \tag{7}$$

The lower bound on T_{opt} over all possible not yet known variables x_{ij} is the most important part of our algorithm. It is developed along the lines used in a related integer polynomial minimax approximation problem that appears in a digital filter design (Kodek 1998, 2002) and is given in the following theorem.

Theorem 1: Let T_{opt} be the minimum cycle time corresponding to the optimal solution of the problem (1–3) in which some of the variables are known integers defined by (5). Then T_{opt} is bounded by

$$T_{\text{opt}} \geq \max_{j=k+1,\ldots,n} \left(\frac{c_j + \sum_{i=1}^{m} \frac{s_i'}{t_{ij}} + p_j + q_j}{\sum_{i=1}^{m} \frac{1}{t_{ij}}} \right) \tag{8}$$

where

$$\begin{aligned} p_j &= \sum_{r=k+1}^{n} c_r \min_{i=1,2,\ldots,m} \left(\frac{t_{ir}}{t_{ij}} \right), \\ q_j &= c_k' \min_{i \notin I_k} \left(\frac{t_{ik}}{t_{ij}} \right), \quad j = k+1, \ldots, n. \end{aligned} \tag{9}$$

Proof: Let h be a number that satisfies

$$h \geq \begin{cases} s_i + \sum_{j=1}^{k} t_{ij} x_{ij}^I + \sum_{j=k+1}^{n} t_{ij} x_{ij}, & i \in I_k \\ s_i + \sum_{j=1}^{k-1} t_{ij} x_{ij}^I + \sum_{j=k}^{n} t_{ij} x_{ij}, & i \notin I_k. \end{cases} \tag{10}$$

Note that h is a lower bound for T_{opt} if we can prove that (10) holds over all possible not yet known x_{ij}. Using (6) equation (10) is simplified

$$h \geq \begin{cases} s_i' + \sum_{j=k+1}^{n} t_{ij} x_{ij}, & i \in I_k \\ s_i' + \sum_{j=k}^{n} t_{ij} x_{ij}, & i \notin I_k. \end{cases} \tag{11}$$

It follows from (11) that variables x_{ij} can be expressed as

$$x_{ij} \le \frac{h}{t_{ij}} - \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k+1 \\ r \ne j}}^{n} \frac{t_{ir}}{t_{ij}} x_{ir}, \quad i \in I_k,\ j = k+1,\ 1, \ldots, n,$$

$$x_{ij} \le \frac{h}{t_{ij}} - \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k \\ r \ne j}}^{n} \frac{t_{ir}}{t_{ij}} x_{ir}, \quad i \notin I_k,\ j = k, \ldots, n. \tag{12}$$

Adding all x_{ij} by index i and using (2) and (7) gives

$$c_j \le \sum_{i=1}^{m} \frac{h}{t_{ij}} - \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k+1 \\ r \ne j}}^{n} \sum_{i=1}^{m} \frac{t_{ir}}{t_{ij}} x_{ir} - \sum_{i \notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik}, \quad j = k+1, \ldots, n, \tag{13}$$

and the lower bound for h can now be written as

$$h \ge \frac{c_j + \sum_{i=1}^{m} (s'_i / t_{ij}) + \sum_{\substack{r=k+1 \\ r \ne j}}^{n} \sum_{i=1}^{m} (t_{ir} / t_{ij}) x_{ir} + \sum_{i \notin I_k} (t_{ik} / t_{ij}) x_{ik}}{\sum_{i=1}^{m} (1 / t_{ij})}, \quad j = k+1, \ldots, n. \tag{14}$$

All the terms in (14) are positive. This means that h is a lower bound over all variables if the lowest possible values of the terms containing variables x_{ir} and x_{ik} are used. The variables x_{ir} are subject to

$$\sum_{i=1}^{m} x_{ir} = c_r, \quad r = k+1, \ldots, n. \tag{15}$$

It is quite easy to see that the sum containing x_{ir} is bounded by

$$\sum_{\substack{r=k+1 \\ r \ne j}}^{n} \sum_{i=1}^{m} \frac{t_{ir}}{t_{ij}} x_{ir} \ge \sum_{\substack{r=k+1 \\ r \ne j}}^{n} c_r \min_{i=1,2,\ldots,m} \left(\frac{t_{ir}}{t_{ij}} \right) = p_j, \quad j = k+1, \ldots, n, \tag{16}$$

since it is obvious that a minimum is obtained if x_{ir} is given the value c_r for index i that corresponds to the lowest of the factors t_{ir}/t_{ij} while all other x_{ir} are set to zero. Similarly, the variables x_{ik} are subject to

$$\sum_{i \notin I_k} x_{ik} = c'_k, \tag{17}$$

and the sum containing x_{ik} is bounded by

$$\sum_{i \notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik} \ge c'_k \min_{i \notin I_k} \left(\frac{t_{ik}}{t_{ij}} \right) = q_j, \quad j = k+1, \ldots, n. \tag{18}$$

Equations (16) and (18) are used in the definitions (9) and this completes the proof. □

Note that Theorem 1 does not include the lower bound for the case $k = n$. The following trivial lower bound, which is valid for all k, can be used in this case

$$T_{\text{opt}} \ge \max_{i \notin I_k} (s'_i + t_{ik} x_{ik}), \quad k = 1, \ldots, n. \tag{19}$$

Note also that index $j=k$ was not used in the derivation of Theorem 1. It is possible to derive the equivalent of (13) for $j=k$ and the corresponding lower bound. Its contribution to the total lower bound was found to be negligible and it is not used in our algorithm.

By choosing $k=0$, one can use (8–9) to compute the lower bound over all possible integers x_{ij}. Applying this to the example from table 1 gives $T_{\text{opt}} \geq 111.736$, which can be rounded up to 111.80 because T_{opt} must be an integer multiple of 0.1. This lower bound is not far from $T_{\text{opt}} = 112.50$. But there is more — the theorem plays a central role in our algorithm because it eliminates the need to use the simplex algorithm for solving the subproblems within the branch-and-bound process.

5. Application of the lower bound theorem

The usefulness of Theorem 1 is based on the following observation: the problem of finding the all-integer solution that gives the lowest cycle time T_{opt} can be replaced by the problem of finding the all-integer solution that has the lowest lower bound for T_{opt}. Both approaches obviously lead to the same solution since T_{opt} equals its lower bound when all variables x_{ij} are integers.

This observation, however, is not sufficient. A new constraint must be introduced on one of the variables $x_{ik}, i \notin I_k$, at each branch-and-bound iteration. This constraint cannot be made on the basis of Theorem 1 alone and requires additional elaboration.

To see how the lower bound depends on x_{ik}, let us define the parameters $T_L(j,k)$ as

$$T_L(j,k) = \frac{c_j + \sum_{i=1}^{m} (s'_i/t_{ij}) + p_j + \sum_{i \notin I_k} (t_{ik}/t_{ij}) x_{ik}}{\sum_{i=1}^{m} (1/t_{ij})}, \tag{20}$$

where $j = k+1, \ldots, n$, and $k = 1, \ldots, n-1$. $T_L(j,k)$ is simply (14) rewritten in a slightly different way. The Theorem 1 lower bound (8), in which the variables x_{ik} are used explicitly (instead of factors q_j), is now equal to

$$T_{\text{opt}} \geq \max_{j=k+1,\ldots,n} T_L(j,k). \tag{21}$$

This lower bound does not include case $k=n$. This is easily corrected if (19) is included. To simplify notation we first define parameters $T_I(i,k)$ as

$$T_I(i,k) = s'_i + t_{ik} x_{ik}, \quad k = 1, \ldots, n, \tag{22}$$

and define the new lower bound $T_{\text{opt}} \geq T_{\text{LB}}(k)$

$$T_{\text{LB}}(k) = \max\left(\max_{i \notin I_k} T_I(i,k), \max_{j=k+1,\ldots,n} T_L(j,k) \right). \tag{23}$$

$T_{\text{LB}}(k)$ is defined for $k = 1, \ldots, n$ (where $T_L(j,n) = 0$). It includes, $T_I(i,k)$ for all k even if it is strictly needed only for $k=n$. There is a good reason for that because the T_I lower bound sometimes exceeds the T_L lower bound. This can occur when the t_{ij} differ by several orders of magnitude as is the case of example from table 1 where a large positive t_{ij} is used instead of ∞. Although the algorithm works if T_I is used for $k=n$ only, experiments show that it is usually faster if it is used for all k.

The lower bound $T_{\mathrm{LB}}(k)$ (23) is the basis of our algorithm. It is a linear function of the variables $x_{ik}, i \notin I_k$, and, as mentioned before, a new constraint must be introduced on one of them at each branch-and-bound iteration.

Let $i_c, i_c \notin I_k$, be the index of the variable $x_{i_c k}$ that is selected for constraining. Selection of the index i_c is simple — any of the indices $i, i \notin I_k$, can be used as i_c. It is more difficult to find the $x^*_{i_c k}$ that will be used in the branch-and-bound iteration to constrain the selected variable to integers $x^I_{i_c k}$, which are the nearest lower and upper neighbours of $x^*_{i_c k}$. $x^*_{i_c k}$ must be a number that gives the lowest possible lower bound $T_{\mathrm{LB}}(k)$ over all possible values of the not yet known variables $x_{ik}, i \notin I_k$, and $x_{ij}, i = 1, \ldots, m, j = k+1, \ldots, n$. Or in other words, $x^*_{i_c k}$ must be at the global minimum of $T_{\mathrm{LB}}(k)$.

It is important to understand why $x^*_{i_c k}$ must be at the global minimum of $T_{\mathrm{LB}}(k)$. It must be because our algorithm uses the property that $T_{\mathrm{LB}}(k)$ is a linear function of the variables x_{ik} and is therefore also convex. The convex property is crucial for the success of our algorithm since it ensures that every local optimum is also global. The algorithm uses this property by stopping the search along a variable in the branch-and-bound process when $T_{\mathrm{LB}}(k)$ exceeds the current best solution T_u. This, however, can be used only if $x^*_{i_c k}$ is such that $T_{\mathrm{LB}}(k)$ does not decrease when an arbitrary integer is added to $x^*_{i_c k}$. The $x^*_{i_c k}$ at the global minimum certainly satisfies this condition.

A great advantage of using the lower bound comes from the fact that $T_{\mathrm{LB}}(k)$ in (23) depends only on the variables $x_{ik}, i \notin I_k$, and is independent of the remaining variables $x_{ij}, i = 1, \ldots, m, j = k+1, \ldots, n$. This means that the number of variables is significantly reduced in comparison with the general approach (5). Solution of the minimax problem

$$T^*_{\mathrm{LB}}(k) = \min_{\substack{x_{ik} \\ i \notin I_k}} \max\left(\max_{i \notin I_k} T_I(i,k), \max_{j=k+1,\ldots,n} T_L(j,k) \right), \tag{24}$$

$$\sum_{i \notin I_k} x_{ik} = c'_k, \qquad x_{ik} \geq 0, \tag{25}$$

gives the non-negative numbers x^*_{ik} that give the global minimum $T^*_{\mathrm{LB}}(k)$ for a given k.

A complication arises when k changes to $k+1$ because the solution of (24–25) for $k+1$ depends not only on x^*_{ik+1}, but also on x^*_{ik} (through s'_i). The problem is that x^*_{ik} is not necessarily at the global minimum of $T_{\mathrm{LB}}(k+1)$. It is possible that the minimum of (24) for $k+1$ decreases if different x^*_{ik} are used. An error can occur if this is ignored because the algorithm stops the search along a variable if $T_{\mathrm{LB}}(k+1) > T_u$ when in fact a lower value for $T_{\mathrm{LB}}(k+1)$ exists.

This complication is solved by replacing (24) with

$$T^*_{\mathrm{LB}}(k) = \min_{\substack{x_{ik}\, x_{ik+1} \\ i \notin I_k \; i=1,\ldots,m}} \max(T_{\mathrm{LB}}(k), T_{\mathrm{LB}}(k+1)), \tag{26}$$

in cases where k will change to $k+1$. Solving (26) gives the correct values for x^*_{ik} and eliminates the possibility of an error. Additional details about the implementation of (26) are given in Step 6 of the algorithm in section 7.

The minimax problem (24–25) must be solved many times within the branch-and-bound process and it is extremely important to have an efficient method that gives its solution. Most of the computing time in our algorithm is spent on solving this problem. The method that is used to solve it is worth a detailed description.

6. Solving the constrained discrete linear minimax problem

The number of variables x_{ik} in (24–25) is equal to the number of indices $i, i \notin I_k$. Let $m', 1 \le m' \le m$, be this number and let $P(i), i = 1, \ldots, m'$, be the indices not in I_k. Equation (24) contains m' terms T_I and $n-k$ terms T_L. The total number of terms n' is equal to

$$n' = n + m' - k, \qquad m' \le n' \le n + m'. \tag{27}$$

It helps to rewrite (24) using a new index v

$$T^*_{\mathrm{LB}}(k) = \min_{x_{P(i)k}} \max\left(\max_{v=1,\ldots,m'} T_I(P(v), k), \max_{v=m'+1,\ldots,n'} T_L(v', k) \right), \tag{28}$$

where $v' = v + k - m'$. Because of the sum constraint in (25) there are only $m' - 1$ independent variables. Any of the variables $x_{P(u)k}, u = 1, \ldots, m'$, can be expressed in terms of the other $m' - 1$ variables. Notation is simplified by defining the indices $R(i), i = 1, \ldots, m'$

$$R(i) = \begin{cases} P(i), & i = 1, \ldots, u-1 \\ P(i+1), & i = u, \ldots, m'-1 \\ P(u), & i = m'. \end{cases} \tag{29}$$

Note that the indices $R(i)$ are a function of u. A variable $x_{P(u)k} = x_{R(m')k}$ is equal to

$$x_{R(m')k} = c'_k - \sum_{i=1}^{m'-1} x_{R(i)k}, \tag{30}$$

and the minimax problem (24–25) can now be reformulated into a more general form

$$T^*_{\mathrm{LB}}(k) = \min_{x_{R(i)k}} \max_{v=1,\ldots,n'} \left(f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x_{R(i)k} \right) \tag{31}$$

$$\sum_{i=1}^{m'-1} x_{R(i)k} \le c'_k, \quad x_{R(i)k} \ge 0. \tag{32}$$

Definitions of terms f_v and Φ_{vi}, $v = 1, \ldots, n', i = 1, \ldots, m' - 1$, are somewhat tedious though they follow directly from inserting (30) into (20) and (22)

$$f_v = \begin{cases} s'_{R(v)}, & v = 1, \ldots, m'-1 \\ s'_{R(m')} + t_{R(m')k} c'_k, & v = m' \\ \dfrac{c_{v'} + \sum_{r=1}^{m} (s'_r / t_{rv'}) + p_{v'} + (t_{R(m')k} / t_{R(m')v'})\, c'_k}{\sum_{r=1}^{m} (1/t_{rv'})}, & v > m' \end{cases} \tag{33}$$

$$\Phi_{vi} = \begin{cases} t_{R(i)k} \quad \text{if } i = v, & 0 \text{ if } i \neq v, \qquad v = 1, \ldots, m' - 1 \\ -t_{R(m')k}, & i = 1, \ldots, m' - 1, \quad v = m' \\ \dfrac{(t_{R(i)k}/t_{R(i)v'}) - (t_{R(m')k}/t_{R(m')v'})}{\sum_{r=1}^{m} (1/t_{rv'})}, & i = 1, \ldots, m' - 1, \qquad v > m'. \end{cases} \tag{34}$$

The process of solving (31–32) is simplified by the theorem that gives the necessary and sufficient conditions for the variables $x_{R(i)k}, i = 1, \ldots, m' - 1$, that minimize (31). The general version of the theorem is given in Demyanov and Malozemov (1990). It is repeated here in a form that applies to our problem.

Theorem 2: The variables $x_{R(i)k}, i = 1, \ldots, m' - 1$, are the optimal solution of the minimax problem (31–32) if and only if the following holds

$$\min_{z_i} \max_{v \in V_{max}(x)} \sum_{i=1}^{m'-1} \Phi_{vi}(z_i - x_{R(i)k}) = 0, \tag{35}$$

over all numbers $z_i, i = 1, \ldots, m' - 1$, that satisfy

$$\sum_{i=1}^{m'-1} z_i \leq c'_k, \quad z_i \geq 0. \tag{36}$$

The set $V_{\max}(x)$ contains those of the row indices $v, v = 1, \ldots, n'$, at which the maximum E is obtained. That is

$$E = \max_{v=1,\ldots,n'} \left(f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x_{R(i)k} \right) = f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x_{R(i)k}, \quad v \in V_{\max}(x). \tag{37}$$

Only the row indices $v, v \in V_{\max}(x)$, that give the extremal values of the function (37) are used in the Theorem 2. The theorem says that $x_{R(i)k}$ is the optimal solution if there are no numbers z_i for which (36) is lower than zero. To show how this can be used to solve (31–32), let us assume that we have a set of numbers $x_{R(i)k}$ with the corresponding extremal set $V_{\max}(x)$. To see if $x_{R(i)k}$ can be improved, consider the set of equations

$$\sum_{i=1}^{m'-1} \Phi_{vi} \Delta x_{R(i)k} = -\varepsilon_v, \quad v \in V_{\max}(x), \tag{38}$$

where all $\varepsilon_v > 0$. If a solution $\Delta x_{R(i)k}$ of (38) gives numbers $z_i = x_{R(i)k} + \Delta x_{R(i)k}$ that satisfy (36) we have an improved solution

$$E - \varepsilon_v = f_v + \sum_{i=1}^{m'-1} \Phi_{vi}(x_{R(i)k} + \Delta x_{R(i)k}), \quad v \in V_{\max}(x), \tag{39}$$

in which E is reduced to $E - \min \varepsilon_v$. Note that ε_v must be small enough so that the extremal set $V_{\max}(x)$ does not change. If, however, there is no such solution of (38) then $x_{R(i)k}$ is the optimal solution because it satisfies the conditions of Theorem 2.

The difficulty lies in equations (38) because there are typically many more variables than equations. It is not simple to determine if a solution $\Delta x_{R(i)k}$ that gives z_i satisfying (36) exists. What is needed is an iterative procedure that

systematically leads to the optimal solution. A procedure that is used in our algorithm is described in the following steps:

(1) The iteration counter l is set to 1. A starting solution is found by trying $x_{P(i)k} = c'_k$ (the remaining $x_{P(i)k}$ are of course zero) for $i = 1, \ldots, m'$ and the lower bound $T_{\mathrm{LB}}(k)$ is computed for each i. The index $i = i_1$ that gives the lowest $T_{\mathrm{LB}}(k)$ gives the starting solution

$$x^{(1)}_{P(i_1)k} = c'_k, \qquad x^{(1)}_{P(i)k} = 0, \ i \neq i_1. \tag{40}$$

This starting solution is used because the experiments show that it is often optimal.

(2) Find the largest of the variables $x^{(l)}_{P(i)k}, i = 1, \ldots, m'$. Use its index i as u and compute the indices $R(i)$ using (29). This choice of u ensures that all of the variables $x^{(l)}_{R(i)k}$ that appear in (41) can increase without violating the constraints (32). Having $R(i)$ use (33) and (34) to compute the problem parameters f_v and Φ_{vi}.

(3) Compute the terms

$$E_v = f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x^{(l)}_{R(i)k}, \quad v = 1, \ldots, n' \tag{41}$$

and find the row indices v that give the maximum $E = \max E_v$. These indices define the extremal set $V_{\max}(x^{(l)})$. Some of the rows $v, v \in V_{\max}(x^{(l)})$, may have identical f_v and $\Phi_{vi}, i = 1, \ldots, m'$ (because some of the machines M_i may be identical or very similar). Only one instance of identical rows is kept in $V_{\max}(x^{(l)})$. Elimination of identical rows is important because it simplifies the problem and also prevents numerical difficulties.

(4) Let e be the number of extremal rows in $V_{\max}(x^{(l)})$. Go to Step 7 if $e \geq m'$. Find the $e-1$ non-zero variables $x^{(l)}_{R(i)k}$ and define as $N(i), i = 1, \ldots, e-1$, the corresponding indices $R(i)$. The indices $R(i)$ of the remaining variables $x^{(l)}_{R(i)k}$ are defined as $Z(r), r = 1, \ldots, m' - e$. Note that some of the variables $x_{Z(r)k}{}^{(l)}$ may be nonzero. Set the index r to 1.

(5) For each of the non-extremal rows q, $q \notin V_{\max}(x^{(l)})$ (excluding identical extremal rows), write the following system of $e+1$ equations with $e+1$ unknowns

$$\begin{aligned} &\sum_{i=1}^{e-1} \Phi_{vi} \Delta x_{N(i)k} + \Phi_{vi} \Delta x_{Z(r)k} + \varepsilon_{rq} = 0, \quad v \in V_{\max}(x^{(l)}) \\ &\sum_{i=1}^{e-1} \Phi_{vi} \Delta x_{N(i)k} + \Phi_{qi} \Delta x_{Z(r)k} + \varepsilon_{rq} = E - E_q. \end{aligned} \tag{42}$$

It is possible that the system's matrix contains rows or columns of zeros or that there is linear dependence; such systems are ignored. The system is solved otherwise. Solution is ignored if $\varepsilon_{rq} \leq 0$ or if $x^{(l)}_{Z(r)k} = 0$ and $\Delta x_{Z(r)k} < 0$. Otherwise check if the variables $x_{N(i)k} + \Delta x_{N(i)k}$, $i = 1, \ldots, e-1$, and $x_{Z(r)k} + \Delta x_{Z(r)k}$ conform to constraints (32). If they do not, simply multiply ε_{rq} by a factor that forces them to be exactly within (32). Such a factor always exists because the Δx's are linearly proportional to ε_{rq}. Save the lowest $\varepsilon_r = \min \varepsilon_{rq}$ over all q and the corresponding

$\Delta x_{N(i)k}, i = 1, \ldots, e-1$, $\Delta x_{Z(r)k}$. This solution gives an improved minimax approximation $E - \varepsilon_r$ for a given r. It will also add one of the non-extremal rows q to the extremal set $V_{\max}(x^{(l)})$ if ε_r was not obtained by multiplication with a factor because of the (32) constraint.

(6) Improved approximation $E - \varepsilon_r$, if it exists, is the best possible improvement when only a variable $x^{(l)}_{Z(r)k}$ is used. An even better improvement may exist for some of the remaining variables. Index r is therefore incremented by 1 and Step 5 is repeated for $r \leq m' - e$. Go to Step 7 if there are no solutions in Step 5 for any of r. Otherwise save the highest $\varepsilon = \max \varepsilon_r$ and the corresponding $\Delta x_{N(i)k}, i = 1, \ldots, e-1$, $\Delta x_{Z(r)k}$. Compute the new best solution

$$\begin{aligned} x^{(l+1)}_{N(i)k} &= x^{(l)}_{N(i)k} + \Delta x_{N(i)k}, \quad i = 1, \ldots, e-1 \\ x^{(l+1)}_{Z(r)k} &= x^{(l)}_{Z(r)k} + \Delta x_{Z(r)k}, \end{aligned} \tag{43}$$

where the variables not appearing in (43) remain unchanged. This solution gives an improved minimax approximation $E - \varepsilon$ and will also, in most cases, add one of the non-extremal rows q to the extremal set $V_{\max}(x^{(l)})$. Increment the iteration counter $l \leftarrow l + 1$ and go back to Step 2.

(7) Solution is optimal. The optimal lower bound $T^*_{\mathrm{LB}}(k)$ is equal to

$$T^*_{\mathrm{LB}}(k) = E, \tag{44}$$

and the optimal variables $x^*_{P(i)k}$ to

$$x^*_{P(i)k} = \begin{cases} x^{(l)}_{R(i)k}, & i = 1, \ldots, u-1 \\ x^{(l)}_{R(m')k}, & i = u \\ x^{(l)}_{R(i-1)k}, & i = u+1, \ldots, m'. \end{cases} \tag{45}$$

Stop.

Proving formally that this procedure always produces the optimal solution is not too difficult. The proof, however, takes too much space and is beyond the scope of this paper. Note that the number of non-zero variables and the size of the extremal set grow with the iterations. It is possible to show that when Step 5 does not give an improved solution for any of the remaining variables this is equivalent to the fact that a solution of (38) does not exist.

Having the optimal variables $x^*_{P(i)k}, i = 1, \ldots, m'$, it remains to select the one that will be used as the new constraint. This is done by computing the products

$$t_{P(i)k} x^*_{P(i)k}, \quad i = 1, \ldots, m'. \tag{46}$$

The index $P(i)$ that gives the largest product is selected as i_c. The reason for this choice is obvious — the largest of products (46) is most likely to give the largest increase of the lower bound $T_{\mathrm{LB}}(k)$.

7. The algorithm

The algorithm is based on the well-known branch-and-bound method, which is described in detail in many textbooks (e.g. Papadimitrou and Steigliz 1982). We assume that the reader is familiar with this method and continue with the description of the algorithm.

An important part of the branch-and-bound method is the branch-and-bound tree. Each node in the tree represents a subproblem that has some of the variables constrained to integers. The efficient organization of the tree is important. It does not, however, influence the results of the algorithm and will not be discussed here. The algorithm is described in the following steps:

(1) Set $k=0$ and use (8–9) to compute

$$T_L(j,0) = \frac{c_j + \sum_{i=1}^{m} (s'_i/t_{ij})}{\sum_{i=1}^{m} (1/t_{ij})}, \tag{47}$$

for $j = 1, 2, \ldots, m$. Note that (47) does not include q_j because $q_j = 0$ for $k = 0$. Sort the lower bounds $T_L(j,0)$ in the ascending order. The problem parameters t_{ij} and c_j are reordered accordingly. It is assumed from here on that $j=1$ corresponds to the lowest $T_L(j,0)$, $j=2$ to the next higher $T_L(j,0)$, and so on. The reasons for this reformulation of the problem are simple. Indices j will be used in the order $j = 1, 2, \ldots, m$ because this strategy quickly eliminates the indices j that give the lowest contribution to the total lower bound $T_{\mathrm{LB}}(k)$ and at the same time keeps the indices that give the highest contribution to $T_{\mathrm{LB}}(k)$. The lower bound is therefore higher and this can significantly reduce the number of branch-and-bound iterations. Several other strategies for selecting the order of indices j were tested; none performed better over a large class of problems.

(2) Set the current best solution T_u to ∞ (a large positive number). The corresponding variables $x_{ij}^{(u)}$ can be set to anything — they will be replaced by one of the solutions quickly. The index u indicates that T_u is an upper bound on T_{opt}. The alternative is to use some heuristic construction and compute a near-optimal starting solution T_u. We found that this is not really necessary because the algorithm quickly produces good near-optimal solutions.

(3) Create the root node. This is done by making $k = 1$, $m' = m$ (this makes the set I_k empty), and solving the problem (31–32) as described by (35–46). The root node's lower bound is set to $T_{\mathrm{node}} = T^*_{\mathrm{LB}}(1)$ and the node's information is stored in the branch-and-bound tree. For each node the stored information contains the following: T_{node}, index k, the size of set I_k (equal to $m - m'$), indices i in I_k (if any), integer variables $x^I_{ij}, j = 1, \ldots, k$, index i_c, and the non-integer variable $x^*_{i_c k}$. Initialize the branching counter N to zero.

(4) Choose the branching node by searching through the nodes of the branch-and-bound tree. Go to Step 8 if no nodes with $T_{\mathrm{node}} < T_u$ are found or if the tree is empty. Add 1 to the branching counter N and choose the branching node according to the following rule: if N is odd, choose the node with the lowest T_{node}, otherwise choose only among the nodes that contain the largest number of integer variables x^I_{ij} and select the one that has the lowest T_{node}. This branching strategy is a combination of the *lowest lower bound* and *depth first* strategies and is used to get many of the near-optimal solutions as fast as possible. The branching node's information is available for the next steps.

(5) Two subproblems are created from the branching node by fixing the node's variable $x^*_{i_c k}$ to integers

$$x^I_{i_c k} = \lfloor x^*_{i_c k} \rfloor, \tag{48}$$

$$x^I_{i_c k} = \lfloor x^*_{i_c k} \rfloor + 1, \tag{49}$$

where $\lfloor x^*_{i_c k} \rfloor$ is the nearest lower integer to $x^*_{i_c k}$. The integers $x^I_{i_c k}$ must of course conform to (25). If $x^I_{i_c k}$ in (49) does not, discard this subproblem (subproblem (48) is never discarded because $x^*_{i_c k}$ satisfies (32)). The number of non-integer variables x_{ik} is reduced by 1

$$m' \leftarrow m' - 1. \tag{50}$$

If $m' \geq 2$ go to Step 6. Otherwise, there is only one noninteger variable x_{ik} left. Its integer value is already determined because (25) gives

$$x^I_{i_c k} + x^I_{ik} = c'_k, \tag{51}$$

and x^I_{ik} is easily computed. All variables x_{ik} are known integers x^I_{ik}, $i = 1, 2, \ldots, m$. Because of this the index k is incremented as described by (5)

$$k \leftarrow k + 1. \tag{52}$$

The new set I_k is made empty ($m' = m$). If $k \leq n$, go to Step 6. Otherwise, we have a case where all of the subproblem's variables x_{ij} are integer. This is a *complete integer solution* and the cycle time T is simply computed as

$$T = \max_{i=1,2,\ldots,m} \left(s_i + \sum_{j=1}^{n} t_{ij} x^I_{ij} \right). \tag{53}$$

If $T < T_u$, we have a new best solution; the current T_u is set to T and the current best solution $x^{(u)}_{ij}$ is replaced by x^I_{ij}. The branch-and-bound tree is searched and all nodes with $T_{\text{node}} \geq T_u$ are removed from the tree. Go to Step 7.

(6) Each of the non-discarded subproblems from Step 5 is solved. The already known integers x^I_{ij} are taken into account by computing s'_i and c'_k using (6) and (7). The largest of the s'_i is found

$$s'_{\max} = \max_{i=1,\ldots,m} s'_i, \tag{54}$$

and the subproblem is discarded if $s'_{\max} \geq T_u$ because it obviously cannot lead to a better solution. Otherwise (33) and (34) are used to compute f_v and Φ_{vi}. The minimax problem (31–32) is solved as described by (40–46) giving $T^*_{\text{LB}}(k)$ and $x^*_{i_c k}$. The subproblem's lower bound is equal to

$$T^{(\text{new})}_{\text{node}} = \max(T^*_{\text{LB}}(k),\ s'_{\max}). \tag{55}$$

The reason for including $s'_{\max}$ is that $s'_i, i \in I_k,$ are not part of the computation of $T^*_{\text{LB}}(k)$. The algorithm works without this modification but is slower.

As mentioned at the end of Section 5, $T^*_{\text{LB}}(k)$ must be computed differently (using (26) instead of (24)) for nodes with $m' = 2$. Such nodes will undergo a change of k to $k+1$ when used in Step 5. Instead of changing (31–32) to accommodate the more complicated (26), we chose a simple local

search approach here. The $T^*_{\mathrm{LB}}(k)$ is computed using (24) and the resulting $x^*_{i_c k}$ is rounded to the nearest integer $x^I_{i_c k}$. Because of $m' = 2$ all variables x_{ik} are known integers and the corresponding $T^*_{\mathrm{LB}}(k+1)$ can be computed. A simple search along $x^I_{i_c k}$ gives a solution to (26), which is then used in (55) to get $T^{(\mathrm{new})}_{\mathrm{node}}$. The local search approach works well because the starting $x^I_{i_c k}$ is almost always at the global minimum.
The subproblem is discarded if $T^{(\mathrm{new})}_{\mathrm{node}} \geq T_u$. Otherwise its information, containing $T^{(\mathrm{new})}_{\mathrm{node}}$ and $x^*_{i_c k}$, is stored as a new node in the branch-and-bound tree.

(7) The subproblem in the branching node from Step 4 is modified (the root node is an exception — it is simply removed from the branch-and-bound tree and we go to Step 4). The branching subproblem is modified by changing the integer variable x^I_{lk} that was created last. The modification is equal to

$$x^I_{lk} \leftarrow \begin{cases} x^I_{lk} - 1 & \text{if } x^I_{lk} \text{ was created by (48)} \\ x^I_{lk} + 1 & \text{if } x^I_{lk} \text{ was created by (49).} \end{cases} \tag{56}$$

This of course means that each node in the branch-and-bound tree must also contain information about the integer variable that was created last and about the way it was created (either by (48) or (49)). The branching node is removed from the tree if the new $x^I_{lk} < 0$ or if $x^I_{lk} > c'_k$ and we go to Step 4. Otherwise, the modified subproblem is solved exactly as in Step 6. Note that k and m' always remain unchanged and that this subproblem can never be *a complete integer solution*. If $T^{(\mathrm{new})}_{\mathrm{node}} < T_u$ the modified subproblem is stored back into the tree, otherwise it is removed from the tree. Go to Step 4.

(8) The current best solution is the optimal solution. The optimal cycle time T_{opt} is equal to T_u and the optimal variables $x^{(\mathrm{opt})}_{ij}$ are equal to $x^{(u)}_{ij}$. Stop.

8. Experimental results

The algorithm was implemented in a program and tested on many different cases. It is typical of the problem (1–3) that there are often many equivalent optimal solutions. One of the three optimal solutions of the example given in table 1 is presented in table 2. It took less than 0.1 s of computer time to find it. A 2.4-GHz Pentium IV computer was used as a platform for all experiments.

To demonstrate the usefulness of our algorithm for realistic size problems, a test set of four circuit boards was created. The number of different component types n ranges from 10 to 100 and the total number of components per board from 404 to 4040. Description of large problems takes a great amount of space — it is nevertheless needed to allow verification of the results. To conserve space, we created the test boards on the basis of the $m = 3$, $n = 10$ problem from table 1. The additional components were defined by shifting the column information of the original problem. The $n = 20$ problem parameters are defined as

$$c^{(20)}_j = \begin{cases} c_j, & j = 1, 2, \ldots, 10 \\ c_{j-10}, & j = 11, 12, \ldots, 20 \end{cases} \qquad t^{(20)}_{ij} = \begin{cases} t_{ij}, & j = 1, 2, \ldots, 10 \\ t_{i1}, & j = 11 \\ t_{ij-11}, & j = 12, 13, \ldots, 20 \end{cases}, \tag{57}$$

Machine M_i	Allocation x_{ij} of components										Assembly time on machine M_i
	1	2	3	4	5	6	7	8	9	10	
1	162	6	49	30	0	0	0	9	0	0	112.5
2	0	84	0	0	0	0	0	0	7	0	112.4
3	0	0	2	0	22	16	12	0	0	5	111.0
Number c_j of components	162	90	51	30	22	16	12	9	7	5	

Table 2. Optimal solution of the cycle time problem from table 1. The solution was obtained with the algorithm described herein.

where $c_j^{(20)}$, $t_{ij}^{(20)}$ are new parameters and c_j, t_{ij} are values from table 1. Similarly, the $n=40$ and $n=100$ problem parameters are defined as

$$c_j^{(40)} = \begin{cases} c_j^{(20)}, & j = 1,2,\ldots,20 \\ c_{10}^{(20)}, & j = 21 \\ c_{j-21}^{(20)}, & j = 22,23,\ldots,40 \end{cases} \qquad t_{ij}^{(40)} = \begin{cases} t_{ij}^{(20)}, & j = 1,2,\ldots,20 \\ t_{ij-20}^{(20)}, & j = 21,22,\ldots,40 \end{cases} \tag{58}$$

$$c_j^{(100)} = \begin{cases} c_j^{(40)}, & j = 1,2,\ldots,40 \\ c_9^{(40)}, & j = 41,\ c_{10}^{(40)}, j = 42 \\ c_{j-42}^{(40)}, & j = 43,44,\ldots,60 \\ c_8^{(40)}, & j = 61,\ c_9^{(40)}, j = 62 \\ c_{10}^{(40)}, & j = 63 \\ c_{j-63}^{(40)}, & j = 64,65,\ldots,80 \\ c_7^{(40)}, & j = 81,\ c_8^{(40)}, j = 82 \\ c_9^{(40)}, & j = 83,\ c_{10}^{(40)}, j = 84 \\ c_{j-84}^{(40)}, & j = 85,86,\ldots,100 \end{cases} \qquad t_{ij}^{(100)} = \begin{cases} t_{ij}^{(40)}, & j = 1,2,\ldots,40 \\ t_{ij-40}^{(40)}, & j = 41,42,\ldots,80. \\ t_{ij-80}^{(40)}, & j = 81,82,\ldots,100 \end{cases} \tag{59}$$

Table 3 gives the results for the case of $m=3$ machines. The cycle time T_{alg} is obtained with our algorithm, T_{opt} is the optimal cycle time (or its lower bound), and T_{rnd} is the rounded cycle time that was obtained by rounding the non-integer linear programming solution to the nearest integers. The standard program GLPK 4.1 (GNU Linear Programming Kit Version 4.1) was used to compute the linear programming solution.

Computing time was limited to a maximum of 60 s for each of the test problems. This limit was used to demonstrate the simplicity and speed of our algorithm. The results show that the optimal solutions were obtained for $n=10$ and $n=20$ boards. The algorithm did not find optimal solutions for $n=40$ and $n=100$ problems. It did, however, find the lower bounds and near-optimal solutions that are at least as good as the rounded one.

m	n	T_{alg}	T_{opt}	T_{rnd}
3	10	112.5	112.5	113.2
3	20	203.4	203.4	203.8
3	40	443.1	≥ 439.90	443.6
3	100	1241.9	≥ 1201.50	1241.9

Table 3. Results for four boards on the assembly line with $m = 3$ machines.

m	n	T_{alg}	T_{opt}	T_{rnd}
6	10	59.5	≥ 58.90	59.8
6	20	105.0	≥ 104.60	105.1
6	40	224.8	≥ 222.80	225.8
6	100	624.6	≥ 603.70	624.7

Table 4. Results for four boards on the assembly line with $m = 6$ machines.

It is interesting to compare the $n = 20$ optimal solution with the one obtained by the standard integer programming software. Our algorithm produced the solution in 4 s — the GLPK 4.1 needed 1132 s. Neither the GLPK 4.1 nor our algorithm converged within 2 h for $n = 40$ and $n = 100$ problems. Results from table 3 were also compared with the results obtained by the genetic algorithm technique (Ji *et al.* 2001), which were found to be not as good as the rounded non-integer solutions.

Experiments with the same four boards were repeated for the assembly line with six placement machines. The additional three machines are identical to the ones in table 3; this was done to avoid the lengthy description of their placement times. The same 60 s time limit was used and table 4 gives the results. The algorithm and the GLPK 4.1 did not find the optimal cycle time for any of the boards. Still, the near-optimal solutions are not far from the lower bounds and are consistently better than the rounded ones.

An interesting and somewhat unexpected result that follows from our experiments is a good quality of the rounded linear programming solutions. The main reason for this is a relatively large number of components of the same type c_j on our test boards. This makes the non-integer solution less sensitive to rounding. It also makes the allocation problem more difficult from the point of view of our algorithm. The algorithm works better when c_j are small because of the reduced search space. Experiments in tables 3 and 4 can be considered the worst-case scenario.

9. Conclusions

A new algorithm for component allocation problem was presented. The algorithm uses a novel approach quite different from the existing ones. It is simple and typically finds the optimal solution for problems with up to 50–80 variables. For larger problems, it finds the near-optimal solutions that seem to be as good or better than those obtained by other near-optimal methods. It also gives the lower bound on the optimum which can be useful to the user.

Acknowledgement

The authors thank Professor B. Vilfan for providing the formal proof of NP-completeness for the problem (1–3).

References

Ammons, J. C., Carlyle, M., Cranmer, L., Depuy, G., Ellis, K., McGinnis, L. F., Tovey, C. A. and Xu, H., 1997, Component allocation to balance workload in printed circuit card assembly system. *IIE Trans.* **29**, 265–275.

Askin, R. G., Dror, M. and Vakharia, A. J., 1994, Printed circuit card family grouping and component allocation for a multimachine, open-shop assembly cell. *Naval Res. Logist.,* **41**, 587–608.

Brucker, P., 1998, *Scheduling Algorithms,* 2nd ed., pp. 274–307 (Berlin: Springer).

Demyanov, V. F. and Malozemov, V. N., 1990, *Introduction to Minimax*, pp. 113–115 (New York: Dover).

Depuy, G. W., Ammons, J. C. and McGinnis, L. F., 1997, Formulation of a general component allocation model for printed circuit card assembly systems, in Proceedings of the 1997 Industrial Research Conference, Miami, FL, USA, pp. 444–449.

Ji, P., Sze, M. T. and Lee, W. B., 2001, A genetic algorithm of determining cycle time for printed circuit board assembly lines. *Eur. J. Oper. Res.* **128**, 175–184.

Kim, Y. D., Lim, H. G. and Park, M. W., 1996, Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *Eur. J. Oper. Res.*, **91**, 124–143.

Kodek, D. M., 1998, A theoretical limit for finite wordlength FIR digital filters, in Proceedings of the 1998 CISS Conference, Princeton, NJ, USA, pp. 836–841.

Kodek, D. M., 2002, An approximation error lower bound for integer polynomial minimax approximation. *Electrotech. Rev.*, **69**, 266–272.

Papadimitrou, C. H. and Steiglitz, K., 1982, *Combinatorial Optimization*, pp. 433–453, (Englewood Cliffs: Prentice-Hall).

Schtub, A. and Maimon, O. Z., 1992, Role of similarity measures in PCB grouping procedure. *Int. J. Prod. Res.*, **30**, 973–983.

Vilfan, B., 2002, NP-completeness of a certain scheduling problem [in Slovenian]. Internal Report, University of Ljubljana, Faculty of Computer and Information Science, Slovenia.