INT. J. PROD. RES., 1998, VOL. 36, NO. 3, 761–784

A genetic algorithm approach to scheduling PCBs on a single machine

O. Z. MAIMON† and D. BRAHA‡*

This paper addresses the problem of scheduling N printed circuit boards (PCBs) on a single machine equipped with an automatic component interchange mechanism. Assume that the total number of different components required to process all N PCBs is greater than the capacity of the spool. If the requisite components are not on the spool, then one or more component switches must occur before the PCB can be processed. The problem consists of finding the order to schedule the PCBs on the axial insertion machine and the components to place on the spool before each PCB is processed. The performance criterion is to minimize the total number of component switches. This problem is addressed employing a genetic algorithm to search the space of alternative solutions. To evaluate the performance of the GA, a heuristic solution based on a travelling salesman formulation is described. Extensive experiments were carried out for both approaches based on data extracted from industrial scenes.

1. Introduction

1.1. *Background and justification*

Printed-circuit-board (PCB) assembly involves mainly the insertion and soldering of electrical components into printed circuit boards. The interested reader may find a complete description of the relevant printed circuit card and component technologies described in Kear 1987 and Coombs 1988.

The term insertion is used to describe the process of placing a through-hole electrical component onto a PCB so that it leads pass through the correct holes in the board or of placing a surface-mount component onto the board in the required position. There are three methods of insertion: (1) dedicated automatic insertion machines; (2) manual assembly workers, including semiautomatic insertion; and (3) robots. Significant cost savings have resulted from the automation of the process of PCB assembly (Boothroyd 1992). Nevertheless, while automatic insertion machines offer new opportunities for greater productivity, they also cause a variety of planning (prior to production) and control problems which have proved openly resistant to exact solutions. Hence, considering the large capital investment required for their acquisition, careful attention must be paid for both their implementation and utilization.

In this paper the focus is on the planning level prior to production, where decisions are made concerning the operation of an automatic insertion machine. The intent is to develop an efficient procedure for sequencing a series of printed circuit

Revision received January 1997.

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

‡ Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel.

* To whom correspondence should be addressed.

boards on a single machine capable of automatically interchanging electrical components, where the time needed for component switching is significant relative to the PCB assembly time. This is the case with the automatic insertion of axial-lead components (also called Variable Centre Distance components.) VCDs include resistors, capacitors, and diodes within the size limitation of the insertion head.

Automatic axial-lead insertion proceeds as follows. Components are stored on a master spool (a package for holding taped axial-leaded components) in the correct order for insertion. The preparation of each sequenced master spool represents one PCB and generally each different PCB will require the master spool to be set-up in different combination of component types. Thus, each master spool change corresponds to several component switches. Master spools are either made off-line and in advance of the component insertion process (by the automatic component sequencer described below) or on-line with the insertion and sequencing processes combined and carried out on one machine. The off-line sequencing process is done by a component sequencing machine (a sequencer). Axial-lead components are purchased on spools, with one component type on each spool (e.g. a spool of 0·5K resistors). Dispensing spools are loaded manually onto the sequencer for every type of component on the board and, from these, a master spool is created. The sequencer may have from 20 to 240 'heads' which hold a single dispensing spool each. Finally, master spool is loaded manually onto the axial-lead inserter. As mentioned above, these two processes, sequencing and automatic component insertion, may be combined and carried out on one machine. Such a machine, about twice as large as a conventional automatic insertion machine, reduces setup time by bringing inventory to the assembly line. Since components are stored at the assembly line, different PCBs can be assembled, by simply loading a new program for each PCB.

For both types of machines (sequencer or inserter with an internal sequencer), each change of component type (e.g. 0·5K resistors) requires 3–5 min of machine's time, time when the insertion machine must be sitting idle. For example, a component sequencer can handle components at a rate of approximately 10 000–25 000/h or one component every 0·14–0·36 s. It is estimated that this primary set-up activity consumes 35–40% of the machine's usable time. In order to demonstrate the degree of utilization likely to be encountered in problems derived from industrial contexts, consider the input data given in Table 1. If we remove all the components prior to the preparation of each sequenced master spool, as is done by many manufacturers, we derive the following estimates:

Average data drawn from industrial context	Parameter
100 PCBs	Batch size for each PCB type
20 PCBs	Number of PCB types
27 Components	The total number of different component types required for each PCB type
120 seconds	Set-up time for any component type
1 Second	Insertion time for any component
2·2 Components	The number of each component type required by each PCB type.

Table 1. Input characteristics of data drawn from industrial scenarios.

(1) the total set-up activity time is estimated as (set-up time for any component type)×(the total number of different component types required for each PCB type)×(number of PCB types)= $120 \times 27 \times 20 = 64\,800$ s;
(2) the total PCB processing time is estimated as (insertion time for any component)×(The total number of different component types required for each PCB type)×(the number of each component type required by each PCB type)×(batch size for each PCB type)×(number of PCB types)= $1 \times 27 \times 2{\cdot}2 \times 100 \times 20 = 118\,800$ s.

Thus, we conclude that the time associated with changing from one PCB to the next consumes about 35% of the machine's usable time. As a consequence, component switching times figure prominently in PCB assembly planning and control. This motivates us to consider in this paper the problem of finding the PCB sequence and component replacement policy that minimizes the total number of component switches.

Secondary set-up activities include loading the automatic insertion and sequencing programs, and calibrating the automatic insertion device. However, the times needed for these operations are constant and negligible relative to the time associated with changing from one PCB to the next.

1.2. *Problem statement*

Each PCB requires its own set of components that must be re-taped and rolled onto the master spool before any assembly can begin. If some components are common to the PCB being removed from the machine and the PCB being put on the machine, the component loading time may be reduced by leaving the common components on the machine. However, if the requisite components are not on the master spool, one or more component switches must occur. In this event, a cost is incurred which is proportional to the total number of component switches. The total number of component switches depends on the sequence in which to process the PCBs and the component types to switch on the master spool before each PCB is processed. This implies that a lot of machine assembly time will be wasted on component switching unless the total number of component changes is minimized. The objective is to minimize the makespan for the fixed set of PCBs. If it is assumed that processing times are constant, this translates to minimizing the overhead associated with changing over from one PCB to the next.

The problem addressed in this paper arises also in the metal working industry when numerically controlled (NC) machines are used to manufacture parts. In the case of the metal working industry, 'job' is analogous to 'PCB', 'tool' is analogous to 'electrical component', and 'tool magazine' is analogous to 'master spool'. Thus, each of the jobs must be processed on a machine by using the tools placed in its limited capacity tool magazine. Although both terms may be used, we apply the microelectronics terms to our model.

1.3. *Relevant literature review*

A great deal of research has been conducted on job scheduling with sequence dependent set-up time (White and Wilson 1977, Foo and Wangner 1983, Ozden *et al.* 1985, Charles-Owaba and Lambert 1988). However, the added complication of determining the set of components placed on the master spool before each PCB is processed undermines the usefulness of many of these results.

Cunningham and Browne (1986) is the earliest work on PCB sequence and component replacement policy in PCB assembly. The strategy adopted decomposes the problem into two subproblems: (1) decide on the order to schedule the PCBs on the machine; and (2) discover the 'best' succession of spool-changes for this schedule. The first module is solved as follows: (1.1) divided PCBs into 'good' groups of manageable size by establishing a criterion of similarity within groups; (1.2) schedule groups; and (1.3) schedule PCBs within groups. Once the sequence of the PCBs is decided, the main consideration of the second module is to ensure that removal of components that are required subsequently is minimized.

Tang and Denardo (1988a) consider a similar problem arises in the metal working industry, and formulate the problem as mixed-integer linear program. They develop a heuristic approach called the Greedy Perturbation (GP) procedure. The first step consists of finding a short Hamiltonian path on the complete graph with N nodes, each of which represents a job (PCB in our case). The arc length between each pair of nodes (jobs) is approximated by the fewest number of tool switches (component switches in our case) needed by that pair of jobs when processed sequentially. The second step employs a policy called 'Keep Tool Needed Soonest' (KTNS) to determine the tooling decision for any given job schedule. They show that KTNS is optimal for the tool replacement problem. The third step consists of finding a job schedule that may incur fewer number of tool switches than the current best job schedule. The GP procedure finds a feasible solution in 'equilibrium', i.e. better performance cannot be obtained by changing either the job sequence or the tool placement specified by the KTNS policy. In a companion paper, Tang and Denardo (1988b) treat the same problem with a different performance criterion: minimize the number of times at which tools are switched.

Bard (1988) formulates a tool allocation problem arising in flexible manufacturing systems (FMS) as a nonlinear integer program solved with a dual-based relaxation heuristic. This has the effect of decoupling the job sequencing subproblem from the tool loading subproblem.

Lofgren and McGinnis (1986) propose a simple greedy heuristic that considers the 'current' machine set-up, identifies the next Printed Circuit Card (PCC) to run based on the similarity of its requirements to the current set-up, and then determines the components to remove (if any) based on their frequency of use by the remaining PCCs.

In all of the above papers, the authors decouple the PCB sequencing subproblem from the component loading subproblem. Clearly, the strategy of decoupling the PCB sequencing and component switches decisions provides access to a wide variety of Travelling Salesman (TSP) and quadratic assignment (QAP) problem heuristics. The PCB sequencing constitutes a Hamiltonian path of a complete graph with N nodes each of which represents a PCB (as discussed in Tang and Denardo 1988a), and a quadratic assignment heuristic (as discussed in Bard 1988) to assign components to the master spool. The disadvantage of the decomposition strategy is that the PCB sequencing subproblem ignores all the component loading decisions before PCB i and the component loading decisions after PCB j in the PCB sequence. This implies that the shortest Hamiltonian path on the graph is determined in a myopic sense. This motivates us to develop a robust heuristic approach, which can deal with 'look-ahead' consideration of component switches. Furthermore, the above mentioned papers assume that the time needed for start-up is fixed. However, our experience is somewhat different. We have noticed that the time

needed for start-up varies depending on the first PCB to be processed and the components involved. This paper describes a genetic algorithm (GA) technique, which has been developed to address the shortcomings listed above. GAs are those that mimic the underlying genetic process in that entities adapt to their environment in order to survive (Goldberg 1989). The GA manipulates the population of different PCB sequences in such a way that poor sequences fade away and successful sequences continually evolve. GAs have been quite successfully applied to optimization problems like scheduling and travelling salesman problems. Jog *et al.* (1991) present a survey on this topic. Grefenstette *et al.* (1985), Oliver *et al.* (1987), Storer *et al.* (1992), Tate *et al.* (1994), also discuss about using GAs to solve the travelling salesman problems. The TSP is relevant because of its relation to the PCB scheduling problem as shown in the next section.

In the next section, the component switching problem addressed in this paper is formalized. In § 3, we present a GA approach to solve the problem. In § 4, we outline one heuristic approach to solving this problem based on previous work on the same problem. Extensive computational results obtained from testing the GA approach against the heuristic are summarized in § 5. Section 6 presents some conclusions and directions for future research.

2. Model formulation

In our model, there is a set of PCBs to be processed on a single insertion machine equipped with an automatic component interchange device. Each PCB requires a specific set of component types. If the requisite component types are not placed on the master spool, then one or more component switches must occur before the PCB can be assembled. A switch of component type occurs when a component type is removed from the master spool and a different component is inserted on the master spool. The moment in time after processing the nth PCB, but before any components are switched is called instant n. There are two types of decision variables: the sequence in which to process the PCBs, and the set of components placed on the master spool before each PCB is processed. In this section we formulate the problem of determining the PCB sequence and corresponding component loading that minimizes the total number of component switches.

Underlying the formulation are the following assumptions: (1) the master spool can accommodate any combination of component types; (2) each different component type occupies only one component slot on the master spool; (3) the total number of different component types required for any PCB must be less than the master spool capacity; (4) the component interchange mechanism will not depend on the particular instant in which it occurs. This implies that no more time is needed to remove a component type i and insert a component type i' at the same instant than at different instants; (5) only one component type is changed at a time, and component changing times are constant and identical. This implies that the time needed for spool switching at any instant is proportional to the number of component changes on the spool; and (6) the set-up time needed for shutdown is fixed, and is part of the standard operating procedure.

We define the following indices:

i index on components $i = 1,\ldots,M$
j index on PCBs $j = 1,\ldots,N$
n index on PCBs positions (in the sequence) $n = 1,\ldots,N$

In addition, we define the following inputs:

A_j $M \times 1$ component requirement vector whose elements are

A_{ji} $\begin{cases} 1 \text{ if component } i \text{ is needed to produce PCB } j \\ 0 \text{ if not} \end{cases}$

L maximum number of different components that can be on the master spool

Finally, we define the following decision variables:

X_{jn} $\begin{cases} 1 \text{ if PCB } j \text{ is the } n\text{th job in the sequence} \\ 0 \text{ otherwise} \end{cases}$

W_n $M \times 1$ vector that describes the components on the master spool at instant n. The instant n is the epoch in time after processing the nth PCB, and before any components are switched.

Specifically

W_{ni} $\begin{cases} 1 \text{ if component } i \text{ is on the master spool at instant } n \\ 0 \text{ otherwise} \end{cases}$

P_n $M \times 1$ binary vector that describes the component switches that occur at instant n;

specifically,

P_{ni} $\begin{cases} 1 \text{ if component } i \text{ switch occurs at instant } n \\ 0 \text{ otherwise} \end{cases}$

With this notation, the problem may be formulated as follows:

Minimize

$$\sum_{n=1}^{N-1} \mathbf{e}P_n + (\mathbf{e}W_1) \quad (\mathbf{e} \text{ is a } 1 \times M \text{ vector of 1's}) \tag{1}$$

Subject to

$$\mathbf{e}W_n \le L \quad \forall n \tag{2}$$

$$X_{jn}A_j \le W_n \quad \forall j, \forall n \tag{3}$$

$$\sum_{j=1}^{N} X_{jn} = 1 \quad \forall n \tag{4}$$

$$\sum_{n=1}^{N} X_{jn} = 1 \quad \forall j \tag{5}$$

$$P_n \ge W_{n+1} - W_n \quad \forall n = 1, \ldots, N-1 \tag{6}$$

$$P_n \ge 0 \quad \forall n = 1, \ldots, N-1 \tag{7}$$

$$X_{jn} \in \{0,1\}; \quad w_{ni} \in \{0,1\}; \quad P_{ni} \in \{0,1\} \tag{8}$$

The objective function (1) counts the number of switches including the switches needed for start-up. Constraint (2) states that the total number of components to be placed on the spool must be less than the capacity L. Note that the assumption concerning constant interchange times (Assumption 5 above) permits Constraint (2) to be tightened. Consequently, there are exactly L component types on the master spool at each instance. Constraint (3) says that if PCB j is assigned to position

n then all the components needed to product PCB j will be on the spool at instant n. Constraints (4) and (5) stipulate that each PCB is assigned to exactly one position in the sequence. Constraints (6) and (7) state that a switch occurs if a component must be loaded at instant n. Constraint (8) is the integrality constraint.

To examine the complexity of the PCB scheduling problem, let us consider a complete graph, $G(V, E)$, in which we associate a primary node in the set V with each PCB. We create an arc between each pair of primary nodes j and k in the graph. The cost of each such arc is the total number of component switches incurred when this pair of PCBs is processed consecutively. Note that the number of component switches between any two PCBs depends on the component loading decision. In addition, we create an auxiliary node which denotes the content of the master spool at the beginning of the day (at instant $n = 0$), where the machine does not perform any loading or unloading operation. An arc is added to the graph between the auxiliary and primary nodes. The cost associated with such arcs is simply the total number of different component types needed by the PCB associated with the primary node. Then every PCB sequence that begins with the auxiliary node corresponds to a Hamiltonian path on the complete graph (where each 'PCB' is visited exactly once). Moreover, the length of such a Hamiltonian path represents the total number of component switches required by the corresponding PCB sequence. Therefore, finding an optimal PCB sequence amounts to finding the minimum distance Hamiltonian path that begins with the auxiliary node on a complete graph with variable arc lengths. Unfortunately, finding the minimum distance Hamiltonian path is equivalent to solving the TSP which is an NP-complete problem. Thus, we conclude that the PCB scheduling problem (1)–(8) is itself an NP-complete problem. This implies that the difficulty of finding the solution increases as the size of the problem increases. Therefore, to generate meaningful solutions, we are justified in searching for heuristic solution procedures. Much of the remainder of this paper is devoted to a discussion of alternative heuristic procedures for solving the problem.

3. The genetic algorithm approach

Traditional heuristic algorithms construct a single solution and then attempt to improve on that solution using a variety of rules. Both the construction algorithms and the improvement procedures are often complex. This complexity is justified since only a single solution is being manipulated and the procedures must attempt to avoid being stuck at a local extrema. GAs, by contrast, simultaneously maintain a large population of solutions and tend to manipulate each solution in much simpler ways (Holland 1975, Goldberg 1989).

To develop a genetic algorithm, schemes for encoding, decoding, and evaluating a solution are required. Often solutions are encoded as bit streams. Once the scheme is decoded, it must be evaluated.

Genetic algorithms being by randomly generating a population of candidate solutions. Given such a population, a genetic algorithm generates a new candidate solution (population element) by selecting two of the candidate solutions as the parent solutions. This process is termed reproduction. Generally, parents are selected randomly from the population with a bias toward the better candidate solutions. Given two parents, one or more new solutions are generated by taking some characteristics of the solution from the first parent (the father) and some from the second parent (the mother). This process is termed crossover. For example, in genetic algo-

rithms that use binary encoding to represent each possible solution, we might randomly select a crossover bit location, $n(1 < n < N)$. Two child solutions could then be generated. The first child would inherit the first n siting characteristics from the father and the remaining $N - n$ characteristics from the mother. The second child would inherit the first n from the mother and the remaining $N - n$ from the father solution. Finally, once child solutions are generated, genetic algorithms allow characteristics of the solutions to be changed randomly in a process known as mutation. In the binary encoding representation, typically, with a small probability (e.g. 0·01) each bit position is changed from its current value to the opposite value. With this probability of a single bit changing, the probability of a child solution remaining unchanged is $(0{\cdot}99)^6 = 0{\cdot}9415$. With probability $> 0{\cdot}88$, both child solutions would remain unchanged.

Once candidate child solutions have been generated, they are decoded and evaluated. Typically, if a child solution is better than the worst solution in the parent population, the child solution replaces the worst element in the parent population. The process continues until some termination criterion is satisfied. Typical termination criteria include stopping when: (1) the value of the best and worst elements in the parent population are either identical or sufficiently close to each other; (2) the value of the best solution in the parent population has not improved after N_1 successive children have been generated; (3) the value of the average solution in the parent population has not improved after N_2 successive children have been generated; or (4) N_3 children have been generated. Goldberg (1989) provides an excellent introduction to genetic algorithms along with theoretical results suggesting why such relatively simple algorithms should work well.

3.1 *PCB scheduling using GA*

The GAs described by Holland (1975) were ones where the solutions of the problem could be represented as strings where each position was independent from the other. Attempting to implement these GAs on combinatorial problems run immediately into the problem of representation. Thus, much of the work done in implementing GAs for combinatorial problems has centred on the joint problem of representing points in the legal search space (genotypes), and devising appropriate operators, corresponding to the mutation and recombination (crossover).

In the context of the PCB scheduling problem formulated above, each PCB is assigned to exactly one position in the sequence. Thus, one natural encoding scheme is to represent a solution as a permutation of the indices of the PCBs. The decoding scheme would then process the PCBs in the order in which they appear in the permutation. The initial population is randomly generated. The outline of the GA designed for the PCB scheduling problem follows.

(1) Randomly generate an initial population of S sequences, SEQ_i, for $i = 1, 2, \ldots, S$.
(2) For each SEQ_i, compute and save the total number of component switches, F_i, based by the Keep Tool Needed Soonest (KTNS) policy (Tang and Denardo, 1988a).
(3) For each SEQ_i, compute its fitness (or utility), U_i, as follows:

$$U_i = \frac{1}{F_i}.$$

(4) For each sequence, compute its selection probability defined by:

$$P(SEQ_i) = \frac{U_i}{\sum_{j=1}^{S} U_j}.$$

(5) Select sequences from the population via the selection probability distribution and apply genetic operators, crossover and mutation. Replace the existing population with the resulting offspring to form a new population of plausible sequences. If termination criterion is not satisfied, return to Step 2. Otherwise, remove the best performing sequence from the population and use it to schedule the PCBs.

Obtaining feasible PCB schedules (Step 2)

Step 2 of the GA employs an approach known as the Keep Tool Needed Soonest (KTNS) policy (Tang and Denardo 1988a) to determine the total number of component switches required by the PCB sequence. The KTNS guarantees optimality for a given PCB sequence. The KTNS policy can be specified as follows:

- At any instant, insert all the components that are required for the current PCB.
- If one or more components must be inserted and there are no empty slots on the master spool, the components that are kept (not removed) from the master spool are those needed the soonest.

In order to describe this policy more precisely, we introduce two definitions, $\mathbf{I}(i,n)$ and $\mathbf{J}(i,n)$. The set $\mathbf{I}(i,n)$ is the set of all instants after instant n at which component i is needed; and the integer $\mathbf{J}(i,n)$ denotes the first instant after instant n in which component i is needed. More formally,

$$\mathbf{I}(i,n) = \{r : \exists j, r > n, X_{jr} = 1 \text{ and } A_{ji} = 1\},$$
$$\mathbf{J}(i,n) = min\{r : r \in \mathbf{I}(i,n)\}.$$

If $\mathbf{I}(i,n) = \phi$, we set $\mathbf{J}(i,n) = N + 1$. For a given PCB sequence and the conclusion that there are exactly L component types on the master spool at each intance, the KTNS policy can be implemented as follows:

KTNS policy

Step 1. Perform Steps 2–3 N times and set $n = 1$.

Step 2. For every component i, set $W_{in} = 1$ if $X_{jn} = 1$ and $A_{ji} = 1$; otherwise set $W_{in} = 0$.

Step 3. **Case 1** ($n < N$). If $\sum_i W_{in} < L$, find i^* that satisfies for each $i \in \{1, \ldots, M\}, \mathbf{J}(i^*,n) \leq \mathbf{J}(i,n)$. Set $W_{i^*n} = 1$ (that is load i^* on the master spool at instant n). Set $\mathbf{J}(i^*,n) = N + 2$. If $\sum_i W_{in} < L$, repeat Step 3; otherwise continue. **Case 2** ($n = N$). If $\sum_i W_{iN} < L$, find a component i such that $W_{iN} = 0$ and $i \in W_{N-1}$ and set $W_{iN} = 1$. If $\sum_i W_{iN} < L$, repeat Step 3; otherwise continue.

Step 4. Stop if $n = N$ set $n \leftarrow n + 1$ and go to Step 2.

Example 1. In order to demonstrate how this policy might be implemented, we consider the input data given in Table 2 for the case where $N = 4$, $M = 5$ and $L = 3$. Given the PCB sequence (population element) $3 \rightarrow 2 \rightarrow 1 \rightarrow 4$, the first two iterations of the KTNS policy might be implemented as follows: First, we set $n = 1$. Following Step 2, the content of the spool at instant $n = 1$ is set to $W_1 = \{2,4,5\}$

Component/PCB	1	2	3	4
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

Table 2. Example component/PCB requirement matrix.

since all the components in W_1 are required by PCB 3. Since $\sum_i W_{i1} = 3$, we skip Step 3, proceed to Step 4 (set $n = 2$) and repeat Step 2. Following Step 2, the content of the spool at instant $n = 2$ is set to $W_2 = \{1, 3\}$ since all the components in W_2 are required by PCB 2. Since $\sum_i W_{i2} < 3$, we insert component $i^* = 2$ on the spool (i.e. $W_{2,2} = 1$), as is needed at the instant $n = 3$. That is, for each $i \in \{1, \ldots, 5\}$, $\mathbf{J}(2,2) \leq \mathbf{J}(i,2)$. Since now $\sum_i W_{i2} = 3$, we proceed to Step 4 (set $n = 3$) and repeat Step 2. In a similar manner we perform the remaining two iterations ($n = 3,4$). The result is the set of components to be placed on the master spool before each PCB is processed as reported in Fig. 1. Figure 1 also shows that in this example, the PCB sequence $3 \rightarrow 2 \rightarrow 1 \rightarrow 4$ entails 7 component switches (including the switches needed for start-up).

Note that the components on the spool at instant n consist of all the components required by the nth PCB in the sequence. Furthermore, the components to be kept are determined by the components to be removed from the master spool. For example, consider instant 2 where components 1 and 3 must be inserted on the spool. By the KTNS policy, component 2 is kept on the spool (and components 4 and 5 are removed) since it is needed at the next instant.

Reproduction, crossover, and mutation (Step 5)

In Step 5 of the GA, sequences are randomly selected for reproduction via the selection probability distribution determined in Step 4. Therefore, sequences with higher utilities (i.e. lower total component switches) are more likely to be selected than those with lower utilities. Sequences are selected two at a time and generic operators, crossover and mutation, are applied to generate two new offspring sequences. This process continues until the population is completely renewed, at which time the algorithm returns to Step 2. Each generic operator is discussed below.

Crossover is performed in two steps. First, the crossover operation on two sequences SEQ_i and SEQ_j creates two new sequences SEQ_i^1 and SEQ_j^2. The opera-

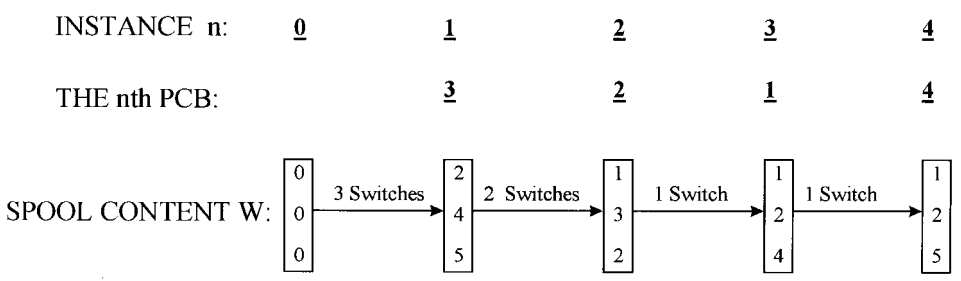

Figure 1. Master spool's content derived by the KTNS policy.

PARENT SOLUTIONS:	**Father**	**Mother**
	(*1*, *2*, *3*, *4*, *5*)	(4, 5, 2, 3, 1)
ONE RANDOMLY SELECTED CROSSOVER POINT:	**Between positions 4 and 5**	
PARENT SOLUTIONS WITH CROSSOVER DELIMITERS:	**Father**	**Mother**
	(*1*, *2*, *3*, *4* | *5*)	(4, 5, 2, 3 | 1)
CONSTRUCTION OF FIRST CHILD SOLUTION:		
a. Copy of Father	(*1*, *2*, *3*, *4* | *5*)	
b. Replace 1st Element:	(*1*, *2*, *3*, *4* | 1)	1 *replaces 5*;
c. Alter 1st Element:	(*5*, *2*, *3*, *4* | 1)	5 *moves to position previously occupied by 1*
SECOND CHILD:	(4, 1, 2, 3 | *5*)	

(*Father characteristics shown in italics to indicate more clearly the inheritance process*)

Figure 2. Example generation of a single child using the modified crossover technique.

tion is performed as follows. Select randomly a crossover location, $n(1 < n < N)$. SEQ_i^1 would inherit the first n siting positions from the father and the remaining $N - n$ positions from the mother. SEQ_j^2 would inherit the first n from the mother and the remaining $N - n$ from the father solution. However, with solutions represented as permutations of the integers 1 to N, the crossover process must be carefully specified to ensure that the child solutions are also permutations. If SEQ_i^1 (or SEQ_j^2) violates this property, the first n positions are altered (Step 2) according to the following rule. Replace a PCB which is already assigned in the remaining $N - n$ positions by the dual PCB determined by the crossover operation. This is illustrated in Fig. 2 for parents characterized by a permutation of the integers from 1 to 5. Note that both children are also permutations of the integers from 1 to 5.

As with most GA's reported in the literature, the mutation operator is infrequently applied (e.g. with a given permutation probability of 0·1) leaving the crossover operator with the most responsibility for conducting the search. If a sequence was selected to undergo mutation, we randomly selected two positions and exchanged the elements in the permutation in these positions.

Termination and convergence criteria

The GA is terminated with either (1) a pre-specified limit on the number of generations being reached; or (2) the difference between the values of the best and worst solutions in the parent population being less than ε times the worst solution (Goldberg 1989). The genetic algorithm can be described as a Markov chain (Davis and Principe 1991). Each state in the Markov chain corresponds to a particular

population of PCB sequences. When the population size and the genetic parameters are held constant during the generations (which is true in our GA), a time-homogeneous Markov chain is obtained with final stationary distribution of the probabilities of the states. The convergence of the GA has been verified on the problems presented in § 5.

4. A spanning tree heuristic for PCB scheduling

4.1. *Heuristic development*

To establish a benchmark by which to evaluate the performance of the GA, a spanning tree heuristic solution is described. The suggested heuristic solution represents the common strategy of decoupling the PCB sequencing subproblem from the component loading subproblem (Bard 1988, Tang and Denardo 1988a). Later in this section, the heuristic approach called the Spanning Tree (ST) procedure used here is compared to the Greedy Perturbation (GP) procedure (Tang and Denardo 1988a) in order to provide rationale for using it over the GP procedure. The ST heuristic consists of three major steps.

Step 1. This generates (as shown in § 2) a complete graph with N primary nodes each of which represent a PCB, and an auxiliary node denoted by 0. The length of each arc, $d(i,j)$, between any two primary nodes (PCBs) is approximated by the fewest number of component switches needed by that pair of PCBs when processed sequentially. The arc length $d(i,j)$ between each pair of primary nodes i and j in the graph is computed as follows. The number of components not required by PCB i equals $L - eA_i$. The number of components needed by PCB j, but not PCB i, equals $(eA_i - A_j^T A_i)$. Hence by holding a component for PCB j in each position not required by PCB i, we conclude that $d(i,j) = max\{0, (eA_i - A_j^T A_i) - (L - eA_i)\}$. Note that for each pair of primary nodes $d(i,j) = d(j,i)$. The arc length between the auxiliary node 0 and the primary node j is simply the total number of different component types needed by PCB j, namely $d(0,j) = eA_j$. Therefore, finding an optimal PCB sequence that begins with the auxiliary node 0 amounts to finding the shortest Hamiltonian path on a complete graph with variable arc lengths, which is equivalent to solving the Travelling Salesman Problem (TSP).

Step 2. This finds a minimum spanning tree to the aforementioned complete graph. A spanning tree is a subgraph that includes all the nodes, is connected, and has no circuits (i.e. for any two nodes, there is *exactly* one path between them). Unlike minimum travelling salesman tours, minimum spanning trees are easy to find in low order polynomial time (Kruskal 1956).

Step 3. This constructs a directed graph by replacing each arc between a pair of primary nodes by a pair of directed arcs, and replacing each arc between the auxiliary and a primary node by a directed arc (see Fig. 3b for the tree of Fig. 3a). Then it finds a reasonably short Hamiltonian tour (PCB sequence) that begins with the auxiliary node 0. To avoid visiting some nodes more than once, we shortcut the tour by proceeding directly to the next unvisited node, as shown in Fig. 3(c). Then, the KTNS policy is employed to determine the component loading decision for this PCB sequence. This procedure is used to generate a pre-specified limit of H different Hamiltonian tours (PCB schedules), and the PCB schedule that entails the fewest number of component switches is selected.

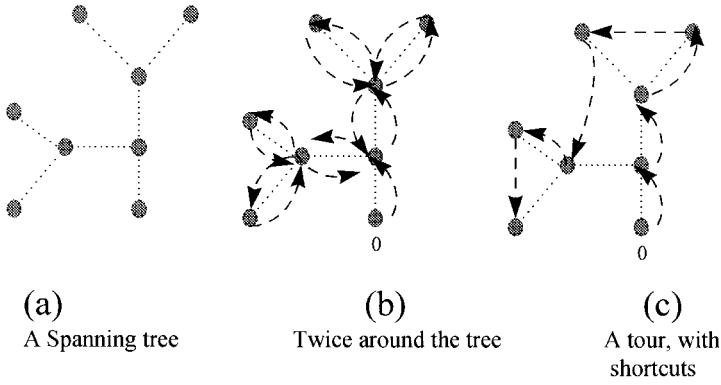

Figure 3. Using 'twice around the tree' algorithm to turn spanning tree into a travelling salesman tour.

Combining these major steps, the ST procedure is specified as follows:

4.1.2. *ST procedure*

Step 1. [Initiation] Construct a complete graph $G(V, E)$ which consists of a set of nodes $V = \{0, 1, 2, \ldots, N\}$ and a set of arcs $E = \{(i,j) | i < j\}$. Assign the fewest number of component switches to each arc (i,j) on the graph (as shown above).

Step 2. [Minimum Spanning Tree] (a) Pick the shortest arc on the graph and put it on the list $\Re$. Remove the arc from the graph. If the selected arc is $(0,j)$ for some node j, then also remove each arc $(0,i)$ for $i \neq j$; (b) If $\Re$ consists of N arcs, then let T be the corresponding spanning tree and go to Step 3; otherwise, continue; (c) Pick the shortest arc on the remaining graph; (d) If that arc forms a cycle with some of the arcs on the list $\Re$, then remove that arc from the graph and go to (c); otherwise, add that arc to the list $\Re$ and got to (b).

Step 3. [Enumeration and Evaluation] Perform Steps (a)–(d) H times: (a) create a new graph from the spanning tree T by replacing each arc (i,j) between each pair of primary nodes by two ordered pairs of nodes $i \rightarrow j$ and $j \rightarrow i$, and each arc $(0,j)$ between the auxiliary node and a primary node by the ordered pair $0 \rightarrow j$; (b) find a Hamiltonian path that begins with the auxiliary node 0 on that graph. If there is more than one route at each node, then select the next node randomly. Avoid visiting some node more than once by proceeding randomly to the next unvisited node; (c) use the KTNS procedure (§ 3) to determine the total number of component switches required by the resulting Hamiltonian path (PCB sequence); (d) pick the PCB sequence that incurs the fewest number of total component switches out of all those H PCB sequences that were generated.

Example 2. We can illustrate the use of the ST heuristic with the matrix of A_{ji} elements for a four PCB/five component problem as shown in Table 2. Let us suppose that $L = 3$. The first step involves the construction of the graph $G(V, E)$. In our example, there are five nodes (including the auxiliary node 0) to be considered. The fewest number of component switches are reported in Table 3. For

i \ j	1	2	3	4
0	3	2	3	1
1	–	1	1	1
2		–	2	0
3			–	0
4				–

Table 3. The fewest number of component switches, $d(i,j)$.

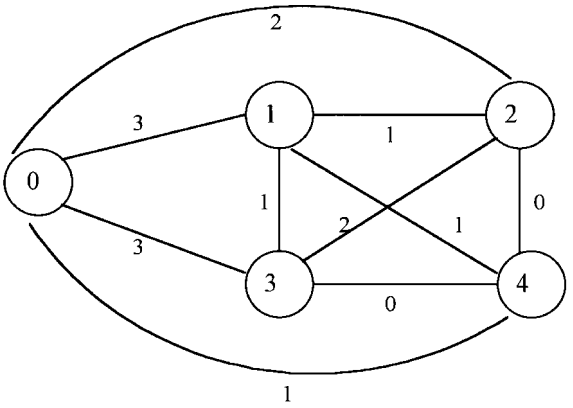

Figure 4. The complete graph derived by step 1 of the ST procedure.

example, $d(1,2) = max\{0,(eA_1 - A_2^T A_1) - (L - eA_1)\} = max\{0,1-(3-3)\} = 1$, and $d(0,3) = eA_3 = 3$. The resulting graph $G(V,E)$ is presented in Fig. 4. The second step consists of finding a minimum spanning tree to the aforementioned graph $G(V,E)$. Figure 5 presents several stages that lead to the minimum spanning tree of $G(V,E)$. The third step initially constructs the directed graph as presented in Fig. 6. Let us suppose that $H = 2$. The first Hamiltonian path that is generated is $0 \rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 2$ (depicted by the dashed line in Fig. 6a), which has length 3. The total number of component switches required by the first PCB sequence as determined by the KTNS policy is 5. The second Hamiltonian path (PCB sequence) that is found is $0 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$ (depicted by the dotted line in Fig. 6b), which has length 4. The total number of component switches required by the second PCB sequence, as determined by the KTNS policy, is 6. Finally, selecting the best PCB sequence out of the two PCB sequences yields the PCB schedule $0 \rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 2$. The set of components placed on the master spool before each PCB is processed, which is determined by the KTNS policy, is reported in Fig. 7.

4.2. *Analysis of the ST heuristic*

Note that $d(i,j)$ ignores all the component decisions before PCB i and the component decisions after PCB j in the PCB sequence. It follows that the length of each Hamiltonian path on the complete graph is a lower bound on the number of component switches required by the corresponding PCB sequence. Thus, the length of the shortest Hamiltonian path is a lower bound on the number of component

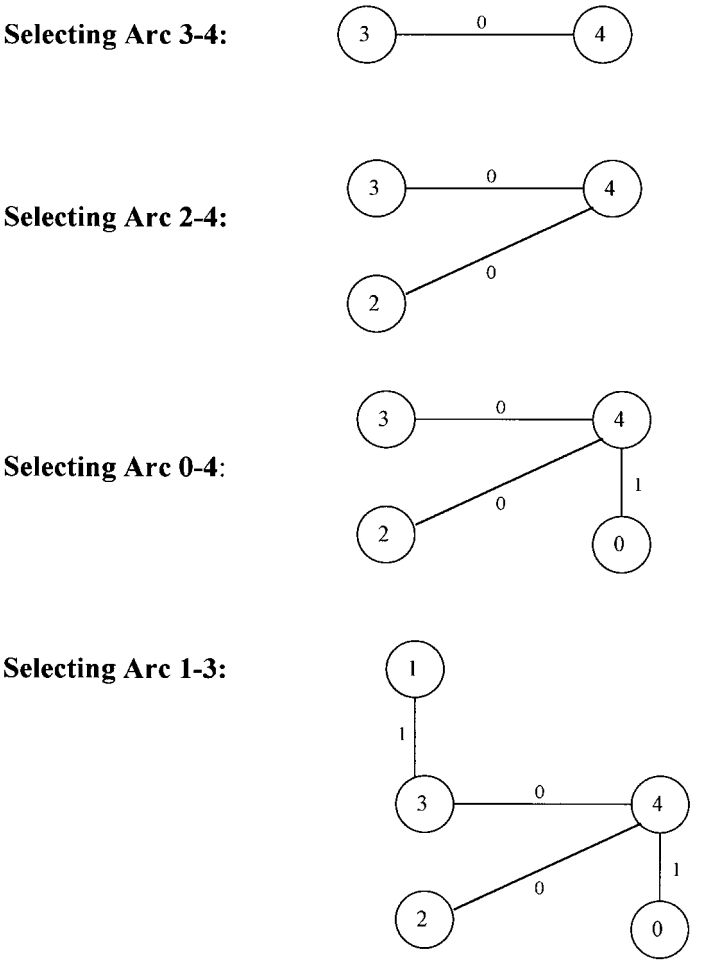

Figure 5. Several stages leading to the minimum spanning tree.

switches entailed by the optimal PCB sequence. Moreover, the length of a minimum spanning tree for a complete graph is a lower bound on the shortest Hamiltonian path. Thus, for graphs obeying the triangle inequality, it is not difficult to see that each resulting tour obtained in Step 3 is at most twice the length of the minimum spanning tree, so it is strictly less than twice the length of the shortest Hamiltonian path on the complete graph. Consequently, each resulting tour obtained in Step 3 is at most twice the length of the number of component switches incurred by the optimal PCB sequence.

4.3. *A comparison with the Greedy Perturbation Procedure* (Denardo and Tang *1988a)*

Finally, we compare the ST heuristic with the Greedy Perturbation (GP) procedure as suggested by Tang and Denardo (1988a). The GP procedure must first be modified in order to include the time needed for start-up. Thus, the GP procedure can be described as follows:

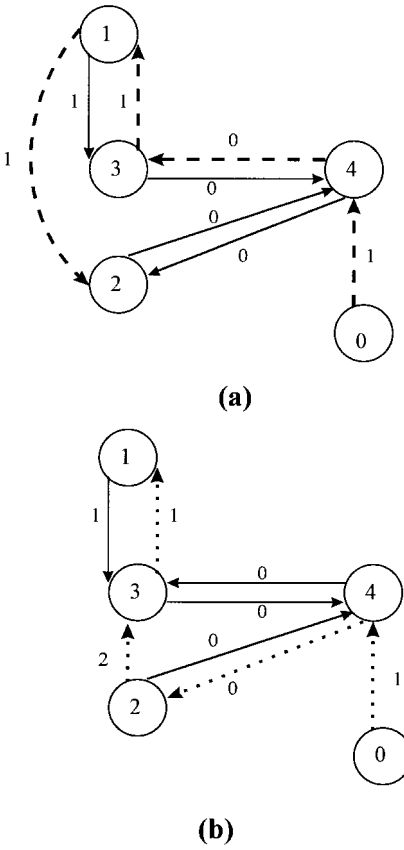

Figure 6. Two PCB sequences derived by step 3 of the ST procedure.

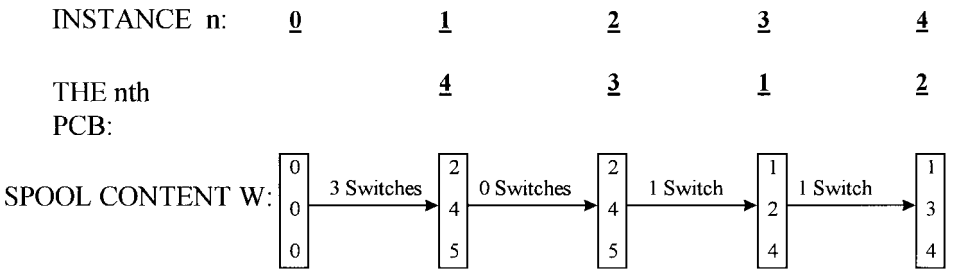

Figure 7. Master spool's content derived by the KTNS policy.

Step 1. This consists of generating a complete graph as delineated by Step 1 of the ST procedure.

Step 2. This consists of finding a short Hamiltonian path that begins with the auxiliary node 0. This is done by developing a greedy procedure which is similar to finding a minimum spanning tree as described by Step 2(d) of the ST procedure, except that in the GP procedure both a tree (node with degree 3) or cycle are excluded.

Step 3. This employs the KTNS policy to determine the component decision for the PCB sequence found in Step 2.

Step 4. This consists of selecting the best PCB sequence out of all those PCB sequences (called perturbed sequences) that can be performed by the same sequence of components that is used to assemble the current best PCB schedule. This best perturbed PCB sequence is set as the current best solution if less component switches are entailed. Improvements continue until either a perturbed PCB sequence cannot be found or a perturbed PCB sequence that entails less component switches than the current PCB schedule is not found. The primary way that the ST heuristic and the GP procedure differ can be described as follows. The GP procedure finds in Steps 2–3 a single PCB schedule and improves it via the Perturbation procedure (Step 4). However, as noted by Denardo and Tang (1988a), the total number of perturbed PCB sequences of any PCB sequence is rather small in practice. This observation suggests that the GP procedure is limited in its ability to search the space of PCB sequences. On the other hand, the ST heuristic finds in Step 2 a minimum spanning tree and generates from it, in Step 3, a prespecified limit of H different PCB schedules. Thus, finding H different PCB schedules in Step 3 (of the ST heuristic) can better explore the space of PCB sequences. The following example applies the GP procedure to the problem considered in Example 2, and shows that the ST heuristic performs better.

Example 3. Consider the matrix of A_{ji} elements for the four PCB/five component problem with $L = 3$ discussed in Example 2 (Table 2). The GP procedure might be implemented as follows. The complete graph which is constructed in Step 1 is given in Fig. 4. Figure 8 presents several stages that lead to a short Hamiltonian path as specified in Step 2. The first Hamiltonian path that is generated is $0 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$, which has length 4. The total number of component switches required by this PCB sequence as determined by the KTNS policy is 6. The set of components placed on the master spool before each PCB is processed is determined by the KTNS policy and is reported in Fig. 9. It can be easily checked that the only PCB sequence that can be performed by the same sequence of components that is used to assemble the current best PCB schedule $0 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ is the perturbed sequence $0 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$. However, since no improvement in the total number of component switches is obtained by the perturbed PCB sequence $0 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$, the GP procedure is terminated. Recall that the ST procedure as shown in Example 2 found a PCB schedule that requires a total number of 5 component switches (as compared to the 6 component switches found by the GP procedure.)

5. Experimental study

In order to investigate the effectiveness of the GA and the ST procedure for the PCB scheduling problem, we generate random input data sets with the following key characteristics:

(1) to analyse the effect of the matrix size on the quality of the solution, the size of the problems analysed ranged from 8 PCBs and 16 components for DATA1, 12 PCBs and 24 components for DATA2 and 16 PCBs and 32 components for DATA3.

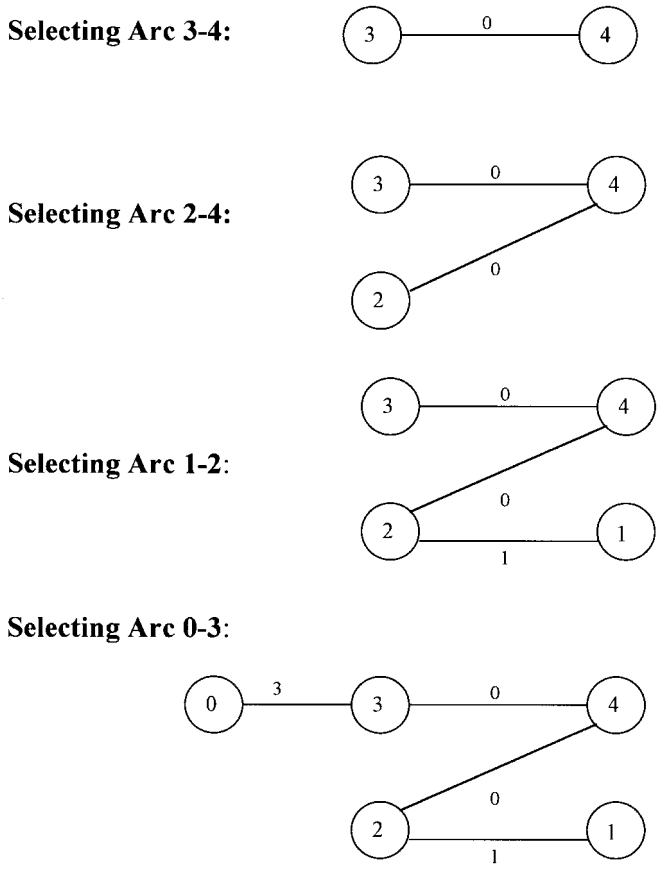

Figure 8. Several stages leading to a short Hamiltonian path.

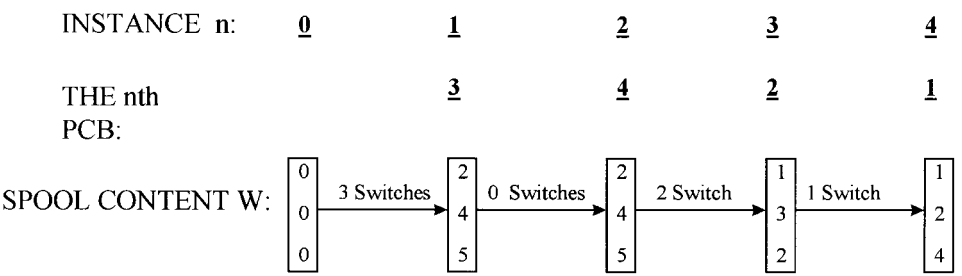

Figure 9. Master spool's content derived by the KTNS policy.

(2) in each instance, the PCB-component matrices were randomly constructed based on two different procedures. In the first case (synthetic generator), we consider that the number of components assigned to each PCB is uniformly distributed over $[1, M]$. In the second case (industrial generator), the PCB-component matrices were randomly generated using data extracted from an industrial manufacturer, who produces 350 000 PCBs per year which involves the insertion and soldering of 23 million electrical components per year into

printed circuit boards. In each matrix, we set the spool capacity on the axial-component sequencing device, L, as the maximum number of different components required to process a PCB.

To test the efficiency and validity of the algorithms, performance is measured by the ratio between the total component switches found by the algorithm and the total number of components required by all PCBs. The latter is an upper bound on the total number of component switches obtained by removing all components from the spool before each PCB is processed. Our industrial experience shows that the process of removing all components prior to the preparation of each sequenced spool holds for most manufacturers.

We conducted the experiments to address the following questions:

(1) what are the optimal values of the parameters involved in the GA and ST procedure?
(2) how does GA compare with ST procedure?
(3) is there reason to conclude that the mean performance resulting from GA exceeds the mean performance resulting from ST procedure?

The proposed algorithms were programmed in C for SUN computers. We addressed these questions via multivariate analysis of variance. We used the ANOVA routines available in SPSS/PC+ (Marija 1994a, b) for statistical analysis of the results. ANOVA deals with the differences between or among treatment means, and imposes no restrictions on the number of treatments. It allows us to study the main effects and interactions between two or more independent variables for a single dependent variable (the performance measure).

5.1. *What are the optimal values of the parameters in the GA and ST procedure?*

The task of formulating GA maps into a parameter selection problem. The main parameters are the population size, mutation probability and termination criterion. The optimal value of these parameters for a particular problem can be obtained by experimentation. In order to analyse the effect of the matrix size on the best setting for each parameter, we conduct a three-factor experiment for each problem size. Performance is measured by the ratio between the total component switches found by the algorithm and the total number of components required by all PCBs. The factors and their chosen levels are listed in Table 4. Our goal in conducting a matrix

Factor	Levels 1	Levels 2
A. Population size	50	100
B. Mutation probability	0·001	0·01
C. Limit on the number of generations†	100	300

† The GA is terminated when: (1) either a prespecified limit on the number of generations to be generated was reached (100 versus 300); or (2) the difference between the values of the best and worst solutions in the parent population was $< 0{\cdot}01\times$ the worst solution.

Table 4. Factors and their levels.

experiment is to determine the best level for each factor. Each treatment combination of three factors is randomly replicated 10× where the PCB-component matrices were randomly constructed based on an industrial generator.

The parameters (including two-parameter interactions) that have the greatest effect on the performance measures are identified by applying analysis of variance and pooling up technique (Ross 1988). The pooling up strategy entails pooling the least column effect with the residual error and F-testing all other columns against the pooled columns. This process repeats itself until all the remaining columns are significant. An alternate strategy for 'pooling up' is 'pooling down'. The pooling down strategy entails pooling all but the largest column effect and F-testing the largest against the remaining pool together. If that column effect is significant, then the next largest is removed from the pool and F-testing is being performed again until some insignificant F ratio is obtained. The pooling up strategy tends to maximize the number of significant factors while the pooling down strategy tends to minimize them. The pooling up strategy tends to increase the alpha type of error, i.e. judging some factors to be significant while, in fact, they are not. On the other hand, the pooling down strategy increases the likelihood of a beta type of error, i.e. judging some significant factors to be non-significant while, in fact, they are. From the practitioner's perspective a beta type of error is less desirable, since once a factor (e.g., termination criterion) has been judged to be insignificant, that factor is excluded from further implementation of the GA, and thus the GA may be prematurely terminated.

For the performance measure defined above, we conclude with one significant parameter (termination criterion as shown in Table 4). The values of F-ratios produced by ANOVA as a function of the problem size are: 2·86 for DATA1, 2·94 for DATA2 and 2·83 for DATA3. The comparison of these F-ratios to $F_{0\cdot1(1,78)}$ indicates that there is a significant difference between the means of the termination criterion treatments.

The coefficient of variation, 5·7% is satisfactorily low. There is little indication from the data that an experimental design more complex than three-factor experiment would be useful in future work of this type.

The statistical analysis partially supports the statement that within reasonable ranges the values of the GA parameters (population size and mutation rate in our case) are not critical. This observation is in partial agreement with Grefenstette (1986).

On the basis of the statistical analysis, we chose the following parameters to run the GA as they gave the best value of performance measure: (a) population size 100; (b) mutation probability 0·01; and (c) either a limit of 300 generations was reached, or the difference between the values of the best and worst solutions in the parent population was $<0\cdot01$ times the worst solution.

Once the GA parameters have been selected, the next step is the analysis of the ST procedure with respect to the pre-specified limit H on the number of different Hamiltonian paths to be generated in Step 3 of the ST procedure. We conducted 75 different experiments with the following characteristics. In order to analyse the effect of matrix size on the best setting for H, we experimented with five different problems for each problem size. For each problem, we employ the ST procedure with varying H (H = 10, 20, 30, 40 and 50) to determine the ratio between the total component switches found by the algorithm and the total number of components required by all

Pre-specified limit H	DATA1 (8×16)	DATA2 (12×24)	DATA3 (16×32)	Total number of problems
10	4	4	1	9
20	1	1	1	3
30	0	0	2	2
40	0	0	1	1
50	0	0	0	0

Table 5. The total number of different Hamiltonian tours explored before a best solution was found.

PCBs. For each problem (for each matrix size), we recorded the pre-specified limit H that was needed before the best solution was found.

Table 5 summarizes our computational experience. In 80% of the runs, the best solution was found for $H \leq 20$. This suggests that even if we had a heuristic procedure that employs larger values of H, we would not be able to save much in terms of performance, since most of the computational effort is devoted to proving optimality of a solution found early in the process. Based on the results shown in Table 5, it seems that the ST procedure is efficient when $H = 30$ (execution times were <19 s) for problems with $(N, M) = (8,16)$, $(12,24)$ and $(16,32)$. Finally, we have noticed that the total number of different Hamiltonian paths that can be generated by the ST procedure in Step 3 depends on the structure of the spanning tree. In practice this number is rather small.

5.2. *How does GA compare with ST procedure?*

We addressed this question via multivariate analysis of variance in Table 6. We conducted a three-factor experiment with replications. Performance is measured by the ratio between the total component switches found by the algorithm and the total number of components required by all PCBs. The three factors in our study are algorithm (GA versus ST procedure), problem size (DATA1 versus DATA2 versus DATA3) and PCB-component matrix generator (synthetic versus industrial generator). Each treatment combination of three factors is randomly replicated 30×. For all three input data sets, all solution times of the GA were <192 s. Therefore, to adequately compare the GA with the ST heuristic, we ran the ST heuristic for, at the most, 192 s of CPU time. Table 6 shows a statistical analysis to determine the prime source of variation in performance. The extremely large F-ratios of the resulting data indicates a real difference between GA and ST procedure as well as a significant effect of the problem size.

Source of variation	F-ratio
Algorithm	53·3†
Problem size	6·71†
PCB-component matrix generator	0·24

† Significant at the 1% level

Table 6. ANOVA for mean performance as a function of algorithm, problem size, and matrix generator.

Problem size	Algorithm	Lower confidence limit	Mean	Upper confidence limit
DATA1	GA	0·3080	0·324	0·3400
	ST procedure	0·3802	0·403	0·4258
DATA2	GA	0·3105	0·323	0·3355
	ST procedure	0·3575	0·380	0·4025
DATA3	GA	0·3165	0·333	0·3495
	ST procedrue	0·3586	0·386	0·4034

Table 7. Using a *t*-distribution to construct confidence intervals on the mean performance as a function of problem size.

5.3. *Is there reason to conclude that the mean performance resulting from GA exceeds the mean performance resulting from ST procedure?*

To answer the third question, a 99% two-sided confidence interval on mean performance (same as before) for each algorithm is constructed. Because Table 6 indicates a real effect of problem size on performance, we conduct statistical analysis for each problem size. From a random sample (based on an industrial generator) of 180 different problems the sample mean performance and sample variance are computed. Table 7 summarizes our computational experience. The resulting confidence intervals for the performance of each algorithm as well as the satisfactorily low coefficient of variation (9–16·2%) strongly support the conclusion that the final PCB scheduling evolved by GA is superior, and therefore, is effective at minimizing the total number of component switches.

6. Summary and directions for future work

In this paper, we have presented a genetic algorithm approach to the component switching problem. The simplicity and robustness made GA attractive especially when it is combined with modern computing power. This approach can deal with 'look-ahead' consideration of component switches, thus transcending the disadvantage of decoupling the PCB sequencing subproblem from the component loading subproblem.

The extensive experiments carried out on data extracted from industrial scenes are very encouraging. The genetic algorithm gives much better results when compared to a benchmark heuristic based on a travelling salesman formulation. The relatively small amount of time needed to find good local solutions (at the most, 3·2 min of CPU time) supports the practicality of using the GA as well as the heuristic for real time control.

The GA described in this paper can be extended to include new crossovers, from which we expected better results. Moreover, augmenting the model to include several machines, PCBs with impending due dates, different performance criteria and splitting batches, is a straightforward matter. Finally, an underlying assumption of the model is that only one PCB is assembled at a time. Based on at least one plant with which the authors are familiar, this is not a bad assumption. However, if PCBs are allowed to be assembled in groups (in which a PCB may be loaded on the machine more than once), the problem becomes the following: find groups of PCBs and a sequence in which to process the groups such that: (1) all PCBs can be produced (i.e. each required component is switched on the spool in at least one group); (2) the number of different component types required by a group of PCBs does not exceed

the spool capacity; and (3) the sum of the number of component switches and PCB loading costs are minimized. In a recent paper, the authors (Daskin *et al.* 1996) address a similar problem where components are completely removed from the machine before a new group of PCBs is loaded. That is, the issue related to the sequencing of groups is not considered. What would be desirable would be to construct a genetic algorithm to search the space of alternative solutions for these both types of problems.

Acknowledgements

Braha's work was partially supported by the Department of Manufacturing Engineering, Boston University, Boston, MA 02215 USA. The financial support is gratefully acknowledged.

References

BARD, S. J., 1988, A heuristic for minimizing the number of tool switching on a flexible machine. *IIE Transactions*, **20**, 382–391.

BOOTHROYD, G., 1992, *Assembly Automation and Product Design* (New York: Marcel Dekker).

CHARLES-OWABA, O. C., and LAMBERT, B. K., 1988, Sequence dependent machine set-up times and similarity of parts: a mathematical model. *IIE Transactions*, **20**, 12–21.

COOMBS, C. F., 1988, *Printed Circuits Handbook* (New York: McGraw-Hill).

CUNNINGHAM, P., and BROWNE, J., 1986, A lisp based heuristic scheduler automatic insertion in electronic assembly. *International Journal of Production Research*, **24**(6), 1395–1408.

DASKIN, M., MAIMON, O., SHTUB, A., and BRAHA, D., 1996, Grouping components in printed circuit board assembly with limited component staging capacity and single card setup: problem characteristics and solution procedures. *International Journal of Production Research*, **35** (6), 1617–1638.

DAVIS, T. E., and PRINCIPE, J. C., 1991, A simulated annealing like convergence theory for the genetic algorithm. *Proceedings of the International Conference on Genetic Algorithms*, pp. 174–181.

FOO, F. C., and WAGNER, J. G., 1983, Set-up times in cyclic and acyclic group technology scheduling systems. *International Journal of Production Research*, **21**, 63–73.

GOLDBERG, D. E., 1989, *Genetic Algorithms in Search Optimization and Machine Learning* (New York: Addison-Wesley).

GREFENSTETTE, J. J., 1986, Optimization of control parameters for genetic algorithms. *IEEE Transactions on Man, Machine and Cybernetics*, **16**(1), 122–128.

GREFENSTETTE, J. J., GOPAL, R., ROSAMITA, B., and VAN GUCHT, D., 1985, Genetic algorithms for the traveling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (Pittsburgh: Carnegie-Mellon University).

HOLLAND, J. H., 1975, *Adaptation in Natural and Artificial Systems* (Ann Arbor: University of Michigan Press).

JOG, P., JUNG, Y. S., and GUCHT, D. V., 1991, Parallel genetic algorithms applied to the travelling salesman problem. *SIAM Journal on Optimization*, pp. 515–529.

KEAR, F. W., 1987, *Printed Circuit Assembly Manufacturing* (New York: Marcel Dekker).

KRUSKAL, J. B., 1956, On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, **7**, 48–50.

LOFGREN, C. B., and MCGINNIS, L. F., 1986, Dynamic scheduling for flexible FCC assembly. *IEEE (ICSMC v.2)*, pp. 1294–1297.

MARIJA, N. J., 1994a, *SPSS/PC+ V2.0 Base Manual* (Chicago, Illinois: SPSS Inc.).

MARIJA, N. J., 1994b, *SPSS/PC+ Advanced Statistics V 2.0* (Chicago, Illinois: SPSS Inc.).

OLIVER, I. M., SMITH, D. J., and HOLLAND, J. R. C., 1987, A study of permutation crossover operators on the traveling salesman problem. *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*.

OZDEN, M., EGBELU, P. J., and IYER, A. V., 1985, Job scheduling in a group technology environment for a single facility. *Computers and Industrial Engineering*, **9**, 67–72.

ROSS, P. J., 1988, *Taguchi Techniques for Quality Engineering* (New York: McGraw-Hill).

STORER, R. H., WU, S. D., and VACCARI, R., 1992, New search spaces for sequencing problems with application to job shop scheduling. *Management Science,* **38**, 1495–1509.

TANG, C. S., and DENARDO, E. V., 1988a, Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. *Operations Research*, **36**(5), 767–777.

TANG, C. S., and DENARDO, E. V., 1988b, Models arising from a flexible manufacturing machine, part I: minimization of the number of switching instants. *Operations Research*, **36**(5), 778–784.

TATE, D. M., TUNASAR, C., and SMITH, A. E., 1994, Genetically improved presequences for Euclidean traveling salesman problems. *Mathematical and Computer Modeling,* **20**(2), 135–143.

WHITE, C. H., and WILSON, R. C., 1977, Sequence-dependent set-up times and job sequencing. *International Journal of Production Research*, **15**, 191–202.