OPERATIONS RESEARCH CENTER

Working Paper

A Multi-Exchange Neighborhood Search Heuristic for an Integrated Clustering and Machine Setup Model for PCB Manufacturing

by
M. J. Magazine
G. G. Polak
D. Sharma

OR 352-01 **March 2001**

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A Multi-Exchange Neighborhood Search Heuristic for an Integrated Clustering and Machine Setup Model for PCB Manufacturing

Michael J. Magazine, Department of QAOM, University of Cincinnati,
Cincinnati, Ohio 45221-0130, telephone (513) 556-7191, Fax (513) 556-5499
michael.magazine@uc.edu

*George G. Polak, MSIS Department, Wright State University,
Dayton, Ohio 45435, telephone (937) 775-3489, Fax (937) 775-3545
george.polak@wright.edu

Dushyant Sharma, Operations Research Center,
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139,
telephone (617) 253-3601, Fax (617) 258-9214
dushyant@mit.edu

* Please address all correspondence to this author.

A Multi-Exchange Neighborhood Search Heuristic for an Integrated Clustering and Machine Setup Model for PCB Manufacturing

Abstract

In the manufacture of printed circuit boards, electronic components are attached to a blank board by one or more pick-and-place machines. Frequent machine setups, though time consuming, can reduce overall processing time. We consider the Integrated Clustering and Machine Setup (ICMS) model, which incorporates this tradeoff between processing time and setup time and seeks to minimize the sum of the two. Solving this model to optimality is intractable for very large-scale instances. We show that ICMS is NP-hard and consequently propose and test a heuristic based on multi-exchange neighborhood search structures. Initial numerical results are very encouraging.

Keywords: Printed circuit board assembly, feeder slot assignment, product clustering, integer programming, computational complexity, heuristics.

1. Introduction

Printed Circuit Board (PCB) production in its entirety is a complex multi-stage operation comprising many interrelated decisions. In each planning period, a manufacturer typically produces a multitude of distinct types of PCB. Each type includes varying numbers of different electronic components, e.g., capacitors, resistors, and microprocessors. For practical reasons, approaches to modeling and optimization have typically treated individual subproblems within an overall production planning framework. Our work concerns an integrated approach to two of these subproblems, that of machine setup and product clustering. Broadly speaking, the former seeks to configure an assembly machine to minimize processing time for a particular production run. The latter seeks sets of PCB jobs that can be efficiently processed together, each under a common machine setup.

There is an extensive literature on the optimization of PCB operations, and we cite a set of works that in some sense spans this field. Ball and Magazine formulated a type of directed postman problem to determine the best sequence of insertion operations and developed an algorithm which is exact under certain conditions, and approximate within constant performance bounds when these conditions are relaxed.[1] Hashiba and Chang formulated an integer program to minimize the number of setups over all sequences of jobs in a PCB assembly shop. In lieu of an exact algorithm, they proposed and tested a three-stage heuristic. [2] Sadiq, Landers and Taylor presented and tested a two-stage heuristic known as the *intelligent slot assignment* algorithm to minimize total

manufacturing time. The first stage sequences jobs to minimize setup time, and given this sequence, the second assigns components to sleeves to minimize processing time.[3] Bard, Clayton and Feo developed a series of heuristics for solving the linked subproblems of component placement sequencing, machine setup, and component retrieval within an iterative procedure.[4]

A series of articles by Moyer and Gupta in the mid-1990s proposed new optimization models to keep pace with rapidly advancing technology in PCB manufacturing.[5-8] First, they developed and tested two heuristics for machine setup, in which component types were assigned to slots in the feederbank to best provide for a specified placement path on the substrate board.[5] Next, they proposed a heuristic they called the Acyclic Assembly Time (AAT) Algorithm for determining the component placement sequence for implementation by a type of assembly machine known as a High Speed Chip Shooter.[6] A subsequent work presented another heuristic for this problem, along with detailed numerical experiments.[7] Finally, they showed how component placement could be significantly speeded by orienting substrate boards at an appropriate angle relative to the placement table.[8]

Jain, Safai and Johnson developed a nonlinear integer model for sequencing jobs in order to minimize setup time, and obtained approximate solutions using a suite of four heuristics. In shop floor testing at Hewlett Packard facilities, these solutions exhibited a tradeoff between setup time and processing time: for large jobs, setup time reduction was surpassed by increased processing time.[9] Assembly operations at Hewlett Packard also motivated a study by Hillier

and Brandeau, who proposed a model for optimally assigning PCB types and components to manual processes as well as to machines. They also developed a heuristic that provides near optimal solutions.[10] Günther, Grunow and Schorling proposed a highly aggregated linear programming model to maximize system throughput in a high mix, low volume facility. To lessen the error of aggregation, the authors used a fuzzy estimation of the number of setups.[11]

There is related research in the broader context of group technology and product clustering. Group technology (GT) takes advantage of similarity within groups of products or parts, with goals that include curtailing machine setups, reducing work-in-process inventory, and improving work center balance. Relevant to PCB manufacturing, Carmon, Maimon and Dar-El proposed a heuristic group set-up (GSU) method for a two-machine PCB assembly process, with an overall objective of increasing throughput.[12] Davis and Selep described the implementation of a "greedy board" GT heuristic to organize PCB board types into clusters, with a primary objective of reducing total setup time.[13] Luzzatto and Perona proposed a multi-criteria heuristic for grouping PCB jobs, which they tested in turn by a simulation model.[14] Ben-Arieh and Chang modified the p-median clustering model to treat p, the number of clusters, as a decision variable. Their objective of minimizing the total measure of dissimilarity among the clusters can be interpreted to be a surrogate for some measure of processing cost or time in the context of manufacturing.[15]

To better deal with this vast body of knowledge, Crama, van de Klundert and Spieksma proposed a common framework for classification of PCB

assembly optimization problems. They defined eight subproblems occurring at one of three levels of a hierarchical planning scheme: assembly shop, assembly lines or manufacturing cells, and individual machines.[16]

Prior to the Integrated Clustering and Machine Setup (ICMS) model of Cohn, Magazine and Polak, PCB Product Clustering and Machine Setup had been treated as separate but linked problems. Unlike conventional clustering approaches that rely on surrogate metrics for similarity of PCB jobs, ICMS forms clusters of PCB jobs by *process commonality*, i.e. prescribing groups of PCB jobs by common machine setup. Thus clustering and setup are solved jointly, avoiding the suboptimization inherent in a hierarchical or sequential solution. Their formulation is a set partitioning integer programming model with an exponential number of columns. To solve it, they applied a branch-and-price algorithm that included a specialized combinatorial search procedure for column generation.[17] Norman devised a set of ICMS test problems and evaluated several classes of heuristics, including clustering by various metrics and genetic algorithms.[18] When the assembler constrains product clustering by fixing the sequence of jobs during the kitting phase of planning without input from the assembly shop floor, as sometimes occurs in practice, Magazine and Polak showed that ICMS can be solved efficiently as a shortest path problem. Moreover they proceeded to analyze the opportunity cost of the organizational barriers that necessitate such procedures.[19]

Section 2 describes PCB manufacturing, defines terminology and lists assumptions. In Section 2 we also consider a special case of ICMS, and analyze

the processing-setup tradeoff. In Section 3, we formulate a combinatorial optimization problem that underlies ICMS. Working from this formulation, we prove that ICMS is NP-hard, motivating development of a heuristic based on multi-exchange neighborhood search structures in Sections 4. Results of initial numerical testing of the heuristic and concluding remarks round out the work.

2. PCB Assembly

2.1 Component Insertion

Ball and Magazine identified six principal steps: (i) *production order release*, from materials requirement planning; (ii) *kitting*, the organizing of kits of appropriate components and the sequential release of jobs; (iii) *prepping*, the operations preparatory to insertion; (iv) *insertion* of components into PCB units; (v) inspection and *soldering* of components; and (vi) *testing* of PCB units for defects, failure, and functionality.[1] In this work our focus falls on steps (ii), (iii) and (iv).

Components are inserted onto a PCB by a pick-and-place machine that uses either through-hole or surface mount technology. In the former, a substrate (blank board) is pre-drilled, and a component pin is inserted into each hole. Solder circuitry is then applied to underside of the substrate. In surface mounting, solder paste is applied to the substrate and each component placed appropriately. The solder paste liquefies in an oven, yielding a finished board when cooled.

Figure 1 illustrates essential features of the pick-and-place machine relevant to ICMS. It has a component magazine known as a feederbank, which is

a linear array of sleeves or slots used to store the various components needed in the manufacture of PCB units. An insertion head moves from its home position at the midpoint of the array to pick an appropriate component from a sleeve and then returns to the home position to insert into a PCB. The board itself moves to the correct position during this head movement and does so quickly enough not to be an issue in this study. These operations are all numerically controlled. Though this technology is neither as new nor as fast as that used for large job fabrication, it is still common today, especially in high voltage or high amperage applications.

[Figure 1 about here.]

2.2 Definitions and Assumptions

We now review the terms and assumptions underlying ICMS. There is a set **K** of jobs, each job comprising a *batch* of b^k units of a board of type k, assembled from a set of **M** types of components in varying amounts. Each type k of board has a distinct component profile, that is, any PCB of type k requires r_i^k of component type i. The position of any component on the board is not relevant to our model because the positioning of the substrate for placement is always quicker than the pick, or retrieval. The pick-and-place machine has a set of **N** uncapacitated sleeves, each of which can hold exactly one type of component. Each delivery and insertion from sleeve j requires d_j seconds and consists of moving the insertion head to that sleeve to pick a component, returning it to the home position, and placing the component on the board. Thus the time to process, i.e., time spent by the pick-and-place machine in component delivery

and insertion, all components of type $i \in \mathbf{M}$ used by job $k \in \mathbf{K}$ from sleeve $j \in \mathbf{N}$ is $c_{ij}^k = b^k d_j r_i^k$.

Next, a *setup* is defined to be an assignment of components by type to feederbank sleeves such that each type of component is assigned to exactly one sleeve and that each sleeve contains exactly one type of component. A machine setup consists of an assignment of component types to sleeves, i.e. a matching from **M** to **N**, and has a constant time of σ. We let *j(i)* denote the sleeve to which component *i* is assigned in a setup. *Total manufacturing time* for a set of jobs consists of the sum of *processing time* and *setup time*.

Given any partition $\wp = \{S_1, S_2, \ldots, S_L\}$ of the set **K** of PCB jobs, each nonempty $S_i \subseteq \mathbf{K}$ is termed a *cluster*. Recall that $S_i \cap S_j = \varnothing$ for $i \neq j$ and $\bigcup_{i=1}^{L} S_i =$ {1, …, *K*}; of course, we must have $1 \leq L \leq K$ for the number of clusters *L*. Each cluster requires a setup taking σ units of time, independent of the choice of assignment, and independent of any prior assignment.

The assumptions below best characterize through-hole insertion technology, but hold reasonably well in the more complicated surface mount environment:

(1) The time to pick and place any component is proportional to the distance of its sleeve from the home position in the feederbank and independent of placement sequence on the substrate board. Because the substrate board is small relative to the feeder bank, positioning for placement is relatively quick, while the pick is the bottleneck operation.

(2) Setup times are independent of the partition of jobs into clusters. Each setup is "full tear-down", and no "partial" setups of the type described by Jain et al are considered. [9] That is, all feeders were removed from the bank at the end of each run, and this activity is quite invariant in time or cost. The full tear-down setup is widely practiced for several reasons: it allows optimal placement of components for each cluster of jobs, it reduces opportunities to make mistakes, and it allows restocking of all feeders at once.

As a consequence of Assumptions (1) and (2), the processing time of a cluster is independent of the sequence of boards and the total manufacturing time is independent of the sequencing of clusters.

(3) The feederbank sleeves are uncapacitated. For many machines, to reload a sleeve is a simple operation requiring very little down time, while other machines are equipped with bulk feeders appropriate for many types of components. In practice, sleeve capacity is quite flexible. The full tear-down mode of setup also affords an opportunity to restock all sleeves so that component "stockouts" are a rare occurrence during processing.

(4) Managerial concerns dictate that each job k belongs to one and only one cluster and cannot be split among several clusters.

Dummy components or sleeves can be employed when the number of components differs from the number of sleeves. If there are more sleeves than types of components, a dummy component indicates an empty sleeve. In the number of component types is larger, then a component that is assigned to a dummy sleeve is actually inserted offline, as described by Hillier and Brandeau.[10]

2.3 Special Case: Single Common Setup

For illustration, consider the special case in which there is only one type of PCB to assemble. Figure 2 shows the "pipe organ" arrangement of the constituent components by type, familiar in material handling, which minimizes total processing time, $b\sum_{i\in N} r_i d_{j(i)}$. (Superscripting by job is superfluous in this case.) That is, the greater the population of a component type, the closer its assigned sleeve should be to the home position. This follows from a classical result of Hardy, Littlewood and Polya, which ensures that $\sum_{i\in N} r_i d_i$ is minimized when nonnegative sequences $\{r_i\}$ and $\{d_i\}$ are arranged monotonically in opposite senses.[20]

[Figure 2 about here.]

The same result applies to a cluster S of jobs given a common machine setup: processing time $\sum_{k\in S} b^k \sum_{i\in N} r_i^k d_{j(i)}$ is minimized by the pipe organ setup, where the component type with the greatest population over the cluster is loaded nearest the home position. We let *f(*S*)* denote minimal processing time for the cluster and $j_*^S(i)$ denote the optimal sleeve assignment for component i$\in$**N** under the pipe organ setup for cluster S.

2.4 The Processing-Setup Tradeoff

A machine setup that minimizes processing time for cluster S need not minimize processing time for any individual job in the cluster. To understand this, suppose that S consists of two jobs, A and B, each possessing a different optimal pipe organ setup. Aggregating their component populations by type can clearly result

in an optimal pipe organ setup for S that differs from that for either constituent job.

Consequently, processing time is *superadditive*, i.e., $f(S) \geq f(T) + f(U)$ given any cluster $S \subseteq \mathbf{K}$ and subsets $T \subseteq S$ and $U \subseteq S$ such that $T \cup U = S$ and $T \cap U = \varnothing$, a fact that we next establish.

Proposition 1. Optimal processing time is a superadditive function on clusters.

Proof. The requirement of any component *i* aggregated over the jobs in cluster S is obtained by simply adding the requirements for T and U, $r_i^S = r_i^T + r_i^U, \ \forall \ i \in \mathbf{N}$. By definition of the optimal processing function,

$$
\begin{aligned}
f(S) &= \sum_{i \in N} d_{j_*^S(i)} r_i^S \\
&= \sum_{i \in N} d_{j_*^S(i)} r_i^T + \sum_i d_{j_*^S(i)} r_i^U \\
&\geq \sum_i d_{j_*^T(i)} r_i^T + \sum_i d_{j_*^U(i)} r_i^U \\
&= f(T) + f(U).
\end{aligned}
$$

The inequality holds because, for any component *i*, sleeve assignment $j_*^T(i)$ and $j_*^U(i)$ are assumed optimal for sub-clusters T and U, respectively, while $j_*^S(i)$ need not be optimal for either. ■

It follows that frequent setups for smaller clusters, though time consuming, can reduce overall processing time. This is the fundamental tradeoff that underlies ICMS.

3. Computational Complexity of ICMS

3.1 The Set Partitioning Integer Programming Model

Each of the 2^K -1 nonempty subsets of the set **K** is a possible cluster of jobs. Consider a binary decision variable x_c for each of these possible clusters, $S_c \subseteq \mathbf{K}$. If $x_c = 1$, then cluster S_c is chosen; if $x_c = 0$, then it is not. Accordingly Cohn et al proposed the following set partitioning integer programming formulation[17]:

ICMS $\quad Minimize \sum_{S_c \subseteq \mathbf{K}} [\sigma + f(S_c)] x_c \qquad (1)$

st

$$\sum_{\{S_c : k \in S_c\}} x_c = 1 \qquad \text{for any board type } k \in \mathbf{K} \qquad (2)$$

$$x_c \in \{0,1\}, \quad \forall c \quad s.t.\ S_c \subseteq K,\ S_c \neq \{\}.$$

Recall that $f(\bullet)$ is defined to be the optimal processing cost function of a cluster, i.e. according to the pipe organ arrangement of the set of components aggregated over all the jobs in the cluster. Since the manufacturing time associated with cluster S_c is the sum of the fixed setup time, σ, plus the processing time associated with an optimal machine setup, the objective function (1) represents the total manufacturing time summed over all clusters in the partition of **K**. The set partitioning constraints (2) ensure that each board type k is included in exactly one cluster S_c. Using *branch-and-price*, a combination of *branch-and-bound* to find integer solutions and *delayed column generation* to manage the exponential number of variables, these authors were able to solve instances of ICMS with as many as 60 PCB jobs and 100 components. (See Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance for a detailed treatment of branch-and-price.[21])

3.2 Formulation as a Combinatorial Optimization Problem

High mix applications of ICMS can involve hundreds of PCB jobs, which at present are beyond the capabilities of any known algorithmic approach. To investigate the computational complexity of ICMS, we formulate an equivalent problem posed purely in terms of sets and set functions.

For any given feasible solution, the corresponding partition $\wp$ of the set **K** is explicit in the objective function (1). On the other hand, the matching of components to sleeves, i.e., the machine setup, within each cluster is implicit in the term *f(•)*. If we reverse these rolls so that each matching is explicitly represented while the choice of partition is implicit, we obtain the following combinatorial optimization problem:

ICMS-CO Minimize $\{\sigma|F| + \sum_{(i,j,k)\in F} c_{ij}^{k} : F \text{ is a feasible component assignment}\}$ (3)

Here we define a feasible set $F = \bigcup_{k\in K} F(k)$, where for each job *k*, *F(k)* is a matching from **M** to **N**, i.e., from component types to sleeves. (Not all the sets *F(k)* need be distinct matchings.) We define |F| to be the cardinality of the feasible set in terms of the number of matchings represented, i.e. the number of distinct matchings among the indexed sets $\{F(k)\}_{k\in K}$. This in turn is the number of machine setups, and also of clusters, since a cluster consists of all products sharing an identical setup. The first term $\sigma|F|$ in (3) therefore represents setup time and the second

$\sum_{(i,j,k)\in F} c_{ij}^k$ processing time; together they constitute total manufacturing time, identical in value to (1) for any partition of **K**.

Now we turn to characterize the computational complexity of ICMS-CO.

Proposition 2. The ICMS-CO (decision) problem is NP-complete.

Proof. The decision version of this problem is "Given the sets *M*, *N*, and *K*, costs $\{c_{ij}^k: i \in M, j \in N, k \in K\}$ and a positive integer *B*, is there a feasible component assignment *F* such that $\sigma |F| + \sum_{(i,j,k)\in F} c_{ij}^k \le B$? First we establish the fact that ICMS-CO is in NP by showing that a *yes* answer to an instance of the decision version of ICMS-CO can be verified in polynomial time. Given a set of triples $\{(i,j,k): i \in M, j \in N, k \in K\}$, we can verify that the subset for any fixed $k \in K$ is a matching from *M* to *N* in O(max{|*M*|, |*N*|}), while to verify that |F| is the number of distinct matchings can be done by comparing each pair of sets in $\{F(k): k \in K\}$ element by element, and thus $O(|K|^2 \min\{|M|,|N|\})$ comparisons. The bound can likewise be verified in a polynomial number of operations.

Now we show that ICMS-CO contains a known NP-complete problem, the Uncapacitated Facilities Location Problem (UFLP), as a special case. Consider a set **L** of candidate facility locations, a set **C** of clients, a "fixed" cost σ_i of placing a facility at location *i*, and a cost c_i^j of servicing the demand by client j from location *i*; see Figure 3. UFLP seeks a subset of locations to service all clients at least total cost, that is:

UFLP $Minimize \sum_{i\in L} \sigma_i y_i + \sum_{i\in L}\sum_{j\in C} c_i^j x_i^j$ (4)

st

$$\sum_{i \in F} x_i^j = 1, \ \forall \ \ j \in \mathbf{L} \tag{5}$$

$$x_i^j \ - \ y_i \leq 0, \ \forall \ i \in \mathbf{L}, \ \forall \ \ j \in \mathbf{C} \tag{6}$$

$$x_i^j, \ y_i \in \{0,1\}, \ \forall \ i \in \mathbf{L}, \ \forall \ \ j \in \mathbf{C}$$

Here

$$y_i \ = \ \begin{cases} 1, \ \textit{if a facility is placed in location } i; \\ 0, \ \textit{otherwise}, \quad \forall\, i \in \mathbf{L}. \end{cases}$$

and

$$x_i^j \ = \ \begin{cases} 1, \ \textit{if client } j \textit{ is serviced from location } i; \\ 0, \ \textit{otherwise}, \quad \forall\, i \in \mathbf{L}, j \in \mathbf{C}. \end{cases}$$

The objective function (4) sums the fixed costs of facilities and the servicing costs from these facilities. Constraint system (5) ensures that each client is serviced from exactly one facility, while (6) forces each servicing decision to be consistent with a facility choice.

[Figure 3 about here.]

UFLP (i.e. the decision version) is known to be NP-complete; see Cornuejols, Fisher and Nemhauser[22] or Karp.[23] Moreover, we do not affect the computational complexity of the problem by using a uniform fixed cost $\sigma_i = \sigma,$ for each location $i \in F$.

Consider now an instance ICMS-CO in which the set **M** of component types corresponds to the set of candidate locations *L* in UFLP, and **K** to the set

of clients **C**. The set of sleeves **N** is {1, …, |M|}. Lastly, we choose the costs $c_{i1}^k = c_i^k$ and $c_{ij}^k = 0$ for j = 2, …, |M|.

In this instance, the objective function of the ICMS-CO for a feasible component assignment F is $\sigma |F| + \sum_{(i,1,k)\in F} c_{i1}^k$. For each $k \in$ **K**, let $i(k)$ be the component such that $(i(k), 1, k) \in F$. Given a feasible component assignment F, we can construct a feasible solution (***x***, ***y***) for the given UFLP instance as:

$x_i = 1$ if $(i, 1, k) \in F$ for some k, and 0 otherwise.

$y_{ij} = 1$ if $i = i(j)$, and 0 otherwise.

The cost of (***x***, ***y***) for the UFLP instance is at most $\sigma |F| + \sum_{(i,1,k)\in F} c_{i1}^k$. Similarly, given a solution (***x***, ***y***) for UFLP, we can create a feasible component assignment F for ICMS-CO such that the objective value of F for ICMS-CO is equal to the cost of (***x***, ***y***) for UFLP. Hence, the optimal values of both the UFLP and the created ICMS-CO instance are equal and the corresponding decision problems are equivalent. This implies that the ICMS-CO (decision problem) is NP-complete.■

Because ICMS-CO is an optimization problem equivalent to ICMS (Garey and Johnson[24]), we have the following result.

Corollary. ICMS is NP-hard.

4. Multi-exchange Neighborhood Search Structures

In this section, we present a neighborhood search based heuristic for ICMS. Neighborhood search algorithms have been successfully applied to solve several hard optimization problems. The primary reasons for the widespread application of neighborhood search techniques in practice are their intuitive appeal, flexibility and ease of implementation, and their excellent empirical results. The standard neighborhood search algorithm for a cost minimization problem P proceeds by (i) constructing a feasible solution, called the *current solution*, $x' \in P$; (ii) defining a neighborhood of x', say $N(x')$, which is the set of solutions of $x \in P$ close to x' in some clearly specified manner; (iii) selecting a neighbor $x'' \in N(x)$, and (iv) replacing the current solution x' by x'' if the cost of x'' is lower than the cost of x'. The algorithm terminates when $N(x)$ has all solutions with the same cost or higher than that of the current solution, in which case the current solution is referred to as a *locally optimal solution* (defined with respect to the given neighborhood). In practice, one usually applies a local improvement algorithm multiple times with different starting solutions, called different *runs*, and selects the best solution obtained among different runs. Neighborhood search has been used in PCB manufacturing earlier by Bard to sequence multiple jobs on a machine to minimize component changes while scheduling different jobs. Using a swap neighborhood for job sequences, he was able to find solutions that were either optimal or close to optimal in reasonable amount of computational time. [25]

The performance of a neighborhood search algorithm critically depends upon the neighborhood structure, the manner in which we define neighbors of a

feasible solution. We present a cyclic exchange neighborhood structure for ICMS. It is based on the cyclic transfer neighborhood structure in Thompson and Orlin. [26] This neighborhood structure has been used effectively earlier for solving set partitioning problems such as Vehicle Routing Problem by Thompson and Psaraftis[27] and Capacitated Minimum Spanning Tree Problem by Ahuja, Orlin and Sharma. [28] We also present a construction heuristic to generate good initial solutions for the neighborhood search.

4.1 Cyclic Exchange Neighborhood for ICMS

We start by presenting some notation to simplify the discussion on cyclic exchange. Recall that we represent a solution to ICMS as a partition $\wp = \{S_1, S_2, \ldots, S_L\}$ of the PCB jobs in the set **K**. Each of the subsets S_i is a cluster of jobs processed together with a single optimal setup. Given a solution S, we use *Cl*(k) to denote the cluster, which contains the board type k in the current solution, i.e., if board type k belongs to S_i, we set *Cl*(k) = i. The optimal processing cost for manufacturing the batches of board types in S_i.is denoted by $f(S_i)$.and that the total cost of the solution S is $c(S) = \sigma L + \sum_{i=1}^{L} f(S_i)$.

We consider two types of exchanges in our neighborhood structure– *cyclic exchanges* and *path exchanges*. We represent a *cyclic exchange* as k_1-k_2-…-k_r-k_1, where the board types k_1, k_2, …, k_r belong to different clusters, that is, $Cl(k_i) \neq Cl(k_j)$ for $i \neq j$. The cyclic exchange k_1-k_2-…-k_r-k_1 represents the following changes: board type k_1 moves from the cluster $Cl(k_1)$ to the cluster $Cl(k_2)$, board type k_2 moves from the cluster $Cl(k_2)$ to the cluster $Cl(k_3)$, and so on, and finally

board type k_r moves from the cluster $Cl(k_r)$ to the cluster $Cl(k_1)$. Let S' denote the new feasible solution. We define the cost of this cyclic exchange as $c(S') - c(S)$. Observe that

$$c(S') - c(S) = \sum_{p=1}^{r} \left(f\left(\{k_{p-1}\} \cup S_{Cl(k_p)} \setminus \{k_p\}\right) - f\left(S_{Cl(k_p)}\right)\right) \qquad (7).$$

where k_0 denotes k_r. In other words, the difference between the costs of new clusters and the previous clusters gives the cost of the cyclic exchange. We call the cyclic exchange *profitable* if $c(S') < c(S)$ and *non-profitable* otherwise. We illustrate the cyclic exchange using the numerical example shown in Figure 4.

[Figure 4 about here.]

We now define a path exchange. We represent a path exchange as k_1-k_2-...-k_r, where the nodes k_1, k_2, ... , k_r belong to different clusters. The path exchange k_1-k_2- ...-k_r represents the following changes: board type k_1 moves from the cluster $Cl(k_1)$ to the cluster $Cl(k_2)$, board type k_2 moves from the cluster $Cl(k_2)$ to the cluster $Cl(k_3)$, and so on, and finally board type k_{r-1} moves from the cluster $Cl(k_{r-1})$ to the cluster $Cl(k_r)$. This exchange is similar to the cyclic exchange except that no board type moves from the cluster $Cl(k_r)$ to the cluster $Cl(k_1)$. In this exchange, the cluster $Cl(k_1)$ loses one board type and the cluster $Cl(k_r)$ gains one board type. Note that if the cluster $Cl(k_1)$ only contained k_1 in S, than the path exchange decreases the number of clusters. Similarly, if $Cl(k_r)$ denotes an empty cluster than the path exchange creates a new cluster containing element k_{r-1}. We do not allow the intermediate clusters to be empty. In

order to take into account the changes in fixed cost due to increase or decrease in the number of clusters by a path exchange, we define the function $\overline{f}(S)$ as:

$$\overline{f}(S) = \begin{cases} f(S) & \text{if } S \neq \varnothing \\ -\sigma & \text{otherwise} \end{cases}$$

If S' denotes the changed solution after this path exchange has been performed, then the cost of this path exchange is

$$c(S')-c(S) = \overline{f}\left(S_{Cl(k_1)} \setminus \{k_1\}\right) - \overline{f}\left(S_{Cl(k_1)}\right) + \sum_{p=2}^{r-1}\left(\overline{f}(\{k_{p-1}\} \cup S_{Cl(k_p)} \setminus \{k_p\}) - \overline{f}(S_{Cl(k_p)})\right) + \overline{f}\left(\{k_{r-1}\} \cup S_{Cl(k_r)}\right) - \overline{f}\left(S_{Cl(k_r)}\right) \quad (8).$$

This path exchange is *profitable* if $c(S') \leq c(S)$ and *non-profitable* otherwise. We illustrate a path exchange using the numerical example shown in Figure 5 below.

[Figure 5 about here.]

Note that the expression (7) for the change in cost due to a cyclic exchange is unaffected if we replace f by $\overline{f}$.

The Neighborhood of a Solution

For a given solution S, we define another solution S' as a *neighbor* of S if S' can be obtained from S by performing a cyclic or path exchange. We define the neighborhood of S as a collection of all solutions that are neighbors of S. The neighborhood based on the cyclic and path exchanges is very large and examining the entire neighborhood to identify a profitable exchange may be an

extremely time-consuming task if we use a brute force method to examine the whole neighborhood. We shall use the concept of improvement graph developed earlier in Ahuja et al[28] to suggest a heuristic that does not enumerate the entire neighborhood but is quite effective in practice.

Improvement Graph for multi-exchange exchange neighborhood

We use S = $\{S_1, \ldots, S_L\}$ to denote the current solution and $G = (N, A)$ to denote the corresponding improvement graph. The graph G contains a node for each board type and for each cluster in S. The arc set in G is defined in a manner so that each cyclic or path exchange with respect to S defines a directed cycle in G, and the cost of the cycle equals the cost of the corresponding exchange.

We now explain how to construct the improvement graph G for solution S. In order to simplify the explanation, we shall denote the board types by indices in {1, ..., K} and their corresponding nodes in G by the same indices. We use S_j to refer to the node in G corresponding to cluster S_j in S. Lastly, we assume that the current solution S contains one empty cluster in order to allow for possibility of generating new clusters by path exchanges. For each ordered pair of board types i and j such that $Cl(i) \neq Cl(j)$ we add arc (i, j) in A. An arc (i, j) between two board types in G signifies that board type i leaves the cluster $S_{Cl(i)}$ and joins the cluster $S_{Cl(j)}$ and simultaneously board type j leaves the cluster $S_{Cl(j)}$. We define the cost c_{ij} of arc (i, j) between nodes corresponding to two board types as

$$c_{ij} = \overline{f}(\{i\} \cup S_{Cl(j)} \setminus \{j\}) - \overline{f}(S_{Cl(j)}) \qquad (9)$$

For each ordered pair (i, S_j) of a board type and a cluster such that $Cl(i) \neq j$ we add arc (i, S_j) to A. Such an arc signifies that board type i leaves the cluster

$S_{Cl(i)}$ and joins cluster S_j. We define cost c_{iSj} of arc (i, S_j) between nodes corresponding to a board type and a cluster respectively as:

$$c_{iS_j} = \overline{f}(\{i\} \cup S_j) - \overline{f}(S_j) \tag{10}$$

We call a directed cycle k_1-k_2- ...-k_r-k_1 in the improvement graph *subset-disjoint* if $Cl(k_p) \neq Cl(k_q)$ for $p \neq q$. Similarly we call a directed path of the form k_1-k_2-...-k_{r-1}-S_j a *subset-disjoint* if $Cl(k_p) \neq Cl(k_q)$ for $p \neq q$ and $Cl(k_p) \neq j$ for $p = 1, \ldots, r-1$. Based on the construction of improvement graph G, we state the following result.

Proposition 3. *There is a one-to-one correspondence between cyclic(path) exchanges with respect to the solution S and directed subset-disjoint cycles(paths) in G, and both have the same cost.*

In order to compute the cost of an arc in the improvement graph, we need to evaluate the two expressions on the right hand side in (9) or (10). Computing the values $\overline{f}(\{i\} \cup S_{Cl(j)} \setminus \{j\})$ (or $\overline{f}(\{i\} \cup S_j)$) and $\overline{f}(S_j)$ can be done in O(|N| log|N|) time if we maintain the total demand for each component from the board types in $S_{Cl(j)}$. Note that $|A| \leq |K|^2$, therefore we can determine the cost of all the arcs in $O(|K|^2|N| \log |N|)$ total time. Observe that we have defined our processing cost function for a specific kind of pick-and-place technology but the neighborhood structure defined here can be applied to other such technologies as well. In particular, this neighborhood could be used in any setting where the processing cost of a cluster can be computed using only information about the board types

in that cluster. This would not be the case, e.g., for sequence dependent processing and setup costs.

Enumerative Strategy for Negative Subset-Disjoint Cycles and Paths

Although there are several efficient algorithms to identify negative (directed) cycles, finding a negative subset-disjoint cycle (path) can be shown to be an NP-complete problem. We shall present an exact algorithm for finding subset disjoint cycles and paths that has an exponential worst case complexity but is very effective empirically.

Our algorithm proceeds by systematically enumerating subset disjoint paths (cycles) of increasing length starting with canonical paths of length 1, i.e. the arcs in *G*. The primary motivation for this technique is that most of the improving exchanges that are identified during the neighborhood search are short, therefore we can identify them early during the enumeration without significant computational overhead. However, note that even for paths with small length, for example 5, enumeration might require examining $O(K^5)$ paths in general. In order to counter this problem we use a property related to the *sub-paths* associated with a negative cost cycle or path. Given a cycle k_1-k_2-…-k_r-k_1 we call a path k_1-k_2-…-k_j, $j \leq r$ as k_1 - k_j *sub-path* in the cycle. The following proposition has been found to be substantially useful in restricting the number of paths examined in practice:

Proposition 4. *Given a negative cost subset disjoint cycle* $C = k_1\text{-}k_2\text{-}\ldots\text{-}k_r\text{-}k_1$, *there exists a node* k_i *in* $k_1, k_2, \ldots, k_r$ *such that each* $k_i\text{-}k_j$ *sub-path starting at* k_i *in the cycle is a negative cost path.*

Proof: Suppose node k_1 does not satisfy this property. Consider the k_1 - k_i sub-path with the largest cost in the cycle, such that k_1 - k_i is longest of all paths with the largest cost. Note that the cost of cycle C, Cost(C), is equal to the cost of subpath k_1 - k_i, Cost(k_1 - k_i) plus the cost of subpath k_l - k_1, Cost(k_l - k_1). Therefore, Cost(k_1-k_i) + Cost(k_l-k_1) < 0. Also Cost(k_l - t_j) < 0 for all j=i+1,...,r,1 because k_1 - k_i is the *longest path* with largest cost. Further since Cost(k_1 - k_p) ≤ Cost(k_1 - k_i) for all p = 2, ..., i -1, Cost(k_l - k_1) + Cost(k_1 - k_p) < 0. In other words, all k_l - k_j sub-paths have negative cost. ■

Using Proposition 4, we only need to enumerate subset disjoint paths of negative cost in our algorithm. Figure 6 gives a formal statement of our negative cost subset disjoint path (cycle) finding algorithm.

[Figure 6 about here.]

Our empirical investigations revealed that the total number of subset disjoint paths generated by the cycle-path finding routine are linear in the number of boards.

4.2 Creating an Initial Solution

The quality of starting solution plays a big role in the success of a neighborhood search solution. Starting a neighborhood search many times with different feasible solutions has been found to be effective in several combinatorial optimization problems, see Festa and Resende.[29] We thus need a mechanism

to generate multiple good feasible solutions for ICMS, and for this purpose employ a maximum savings heuristic. The heuristic starts with |K| clusters each containing a single board type. At each iteration, the algorithm identifies a pair of clusters such that the savings in processing cost achieved by the join operation are maximized. We randomize this heuristic by determining the p most profitable join operations and choosing one from them randomly. Since at each step it performs one of the *p* most profitable join operations, the solution obtained is generally a good tree. In our investigations, we used $p = 2$. Note that after any iteration we only need to update the benefit of joining the newly formed cluster with other clusters if we maintain all the benefit values in a table. Therefore the computational work required at each iteration is at most $O(|K||N| \log|N|)$. Constructing the initial table of benefit values takes $O(|K|^2|N| \log|N|)$ time, and hence the total time needed to compute a solution is at most $O(|K|^2|N| \log|N|)$.

5. Numerical Results

We have begun to test the multi-exchange neighborhood search heuristic on several different PCB planning scenarios, including board profiles, batch sizes, and setup costs devised by Norman.[18] The production environment in each scenario was classified as *low mix* (K = 16 board types), *medium mix* (K = 32 board types), or *high mix* (K = 60 board types). Component variety was either *low* (N = 16 types) or *high* (N = 100 types). Board profiles were classified as *flat*, *60/40*, or *80/20*. A flat profile indicates that all component requirements were sampled from a single uniform distribution. A 60/40 profile indicates that 60% of the component requirements were sampled from one uniform distribution and

40% from another, while an 80/20 profile indicates that 80% of the component requirements were sampled from one uniform distribution and 20% from another. Each setup time was chosen to be the simple average of a "low" value at which a selected clustering heuristic placed each job in a cluster by itself, and a "high" value at which the heuristic placed all jobs in a single cluster. Table 1 displays results for twelve test problems obtained on a desktop PC.

[Table 1 about here.]

6. Future Directions and Conclusions

The Integrated Clustering and Machine Setup Problem is an integer programming model that incorporates two fundamental problems in PCB production planning. Optimal solutions to many large scale industrial instances are out the reach of even a state-of-the-art branch-and-price algorithmic approach. We have shown that ICMS is NP-hard, and consequently have proposed a heuristic based on multi-exchange neighborhood search structures. During initial testing, this heuristic has in each instance obtained a solution as good or better than the best heuristic solution reported to date. Moreover our heuristic found an optimal solution to ICMS for each case in which one is known.

Clustering problems arise in many manufacturing and logistical operations, which exhibit a similar tradeoff between processing and setups and moreover have underlying combinatorial structures such as matchings and paths. In railroad freight transportation, for instance, railcars with several different origin-destination pairs of nodes in a network are clustered together in a *block* in *classification* yards (see Barnhart, Jin, and Vance[30]). The block itself is characterized by an origin-destination pair that may differ from that pair associated with any given constituent car. Problems of this type present an opportunity for research on set partitioning models that generalize ICMS. Based on our experience, multi-exchange neighborhood search heuristics are likely to present a promising approach to such large scale instances of such problems.

References

1. Ball, M.O. and M.J. Magazine, "Sequencing of Insertions in Printed Circuit Board Assembly", *Operations Research* **36**(2) (1988),192-201.

2. Hashiba, S. and T.-C. Chang, "PCB Assembly Reduction Using Group Technology", *Computers and Industrial Engineering* **21**(1-4) (1991), 453-457

3. Sadiq, M., T.L. Landers and G.D. Taylor, "A heuristic algorithm for minimizing total production time for a sequence of jobs on a surface mount placement machine", *International Journal of Production Research* **31**(6) (1993), 1327-1341.

4. Bard, J.F., R.W. Clayton and T.A. Feo, "Machine Setup and Component Placement in Printed Circuit Board assembly", *International Journal of Flexible Manufacturing Systems* **6** (1994), 5-31.

5. Moyer, L. and Gupta, S. M., "SMT Feeder Slot Assignment for Predetermined Component Placement Paths", Journal of Electronics Manufacturing **6**(3) (1996), 173-192.

6. Moyer, L. and Gupta, S. M., "Simultaneous Component Sequencing and Feeder Assignment for High Speed Chip Shooter Machines", *Journal of Electronics Manufacturing*, **6**(4) (1996), 271-305.

7. Moyer, L. and Gupta, S. M., "An Efficient Assembly Sequencing Heuristic for Printed Circuit Board configurations", *Journal of Electronics Manufacturing*, **7**(2) (1997), 143-160.

8. Moyer, L. and Gupta, S. M., "Development of the Surface Mount Assembly Process Through and Angular Board Orientation", *International Journal of Production Research*, **36**(7) (1998), 1857-1881.

9. Jain, S., M.E. Johnson and F. Safai, "Implementing Setup Optimization on the Shop Floor", *Operations Research* **43**(6): (1996), 843:851.

10. Hillier, M.S. and M.L. Brandeau, "Optimal Component Assignment and Board Grouping in Printed Circuit Board Manufacturing", *Operations Research* **46**(5) (1998) 675-689.

11. Günther, H.-O., M. Grunow and C. Schorling, "Workload planning in small lot printed circuit board assembly", *OR Spektrum* **19** (1997)147-157.

12. Carmon, T.F., O.Z. Maimon, and E.M. Dar-El, "Group set-up for printed circuit board assembly", *International Journal of Production Research*, **27**(10) (1989),1795-1810.

13. Davis, T. and E. Selep, "Group Technology for High-Mix Printed Circuit Assembly", *IEEE International Electronic Manufacturing Technology Symposium [proceedings]*, 264-269, 1990.

14. Luzzatto, D. and M. Perona, "Cell formation in PCB assembly based on production quantitative data", *European Journal of Operations Research*, **69**(3) (1993),12-329.

15. Ben-Arieh, D. and P.T. Chang, "An Extension to the p-Median Group Technology Problem", *Computers and Operations Research* **21**(2) (1994),119-125.

16. Crama, Y., J. van de Klundert, and F. C.R. Spieksma, "Production Planning Problems in Printed Circuit Board Assembly", GEMME 9925, University of Liege, Liege, Belgium, 1999.

17. Cohn, A.M., M.J. Magazine and G.G.Polak, "An Optimal Algorithm for Integrating Printed Circuit Board Manufacturing Problems", University of Cincinnati QAOM Working Paper, 2001.

18. Norman, S.K., "Heuristic Approaches to Batching Jobs in Printed Circuit Board Assembly", Ph.D. dissertation, QAOM Department, the University of Cincinnati, 2001.

19. Magazine, M.J. and G.G.Polak, "Product Clustering and Machine Setup in PCB Manufacturing and the Impact of Job Release Policy", University of Cincinnati QAOM Working Paper, 2001.

20. Hardy, G.J., Littlewood and G. Polya, *Inequalities*, Second edition, Oxford University Press, 1952.

21. Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs", *Operations Research* **46**(3) (1998), 316-329.

22. Cornuejols, G., M.L. Fisher and G.L. Nemhauser, "Location of Bank Accounts to Maximize Float: An Analytic Study of Exact and Approximate Algorithms", *Management Science* **23**(8) (1977) 789:810.

23. Karp, R.M., "On the Computational Complexity of Combinatorial Problems", Networks **5** (1975) 45-68.

24. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

25. Bard, J.F., "A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine", *IIE Transactions* **20**(4) (1988) 382-391

26. Thompson, P.M. and J. B. Orlin. "The Theory of Cyclic Transfers" M.I.T. Operations Research Center Report OR 200-89, 1989.

27. Thompson, P.M. and H.N. Psaraftis (1993), "Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling", *Operations Research* **41**(5) (1993) 935-946.

28. Ahuja, R.K., J.B. Orlin, and D. Sharma, "Multi-Exchange Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem", in review for *Mathematical Programming*, 2001.

29. Festa, P. and M.G.C. Resende, "GRASP: An annotated bibliography", To appear in *Essays and Surveys on Metaheuristics*. P. Hansen and C.C. Ribeiro, eds, Kluwer Academic Publishers, 2001.

30. Barnhart, C., H. Jin and P.H. Vance, "Railroad Blocking: A Network Design Approach", *Operations Research* **48**(4) (2000) 603-614.

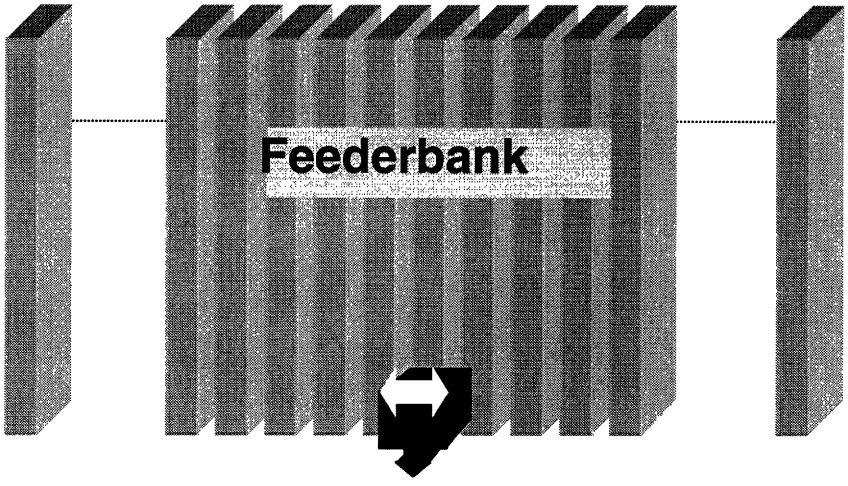

Figure 1. A pick-and-place machine. The insertion head is shown in home position at the midpoint of the feederbank.

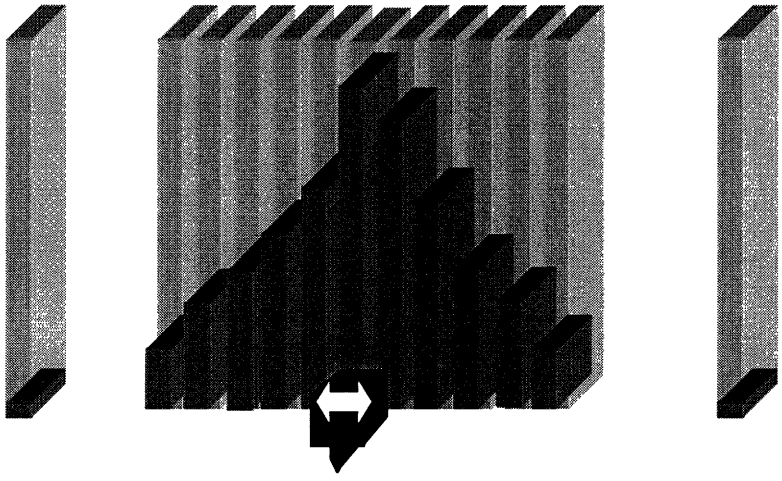

Figure 2. An optimal machine setup, i.e., assignment of component types to feeder sleeves. The greater the population of a type, the closer its assigned sleeve to the insertion head's home position.

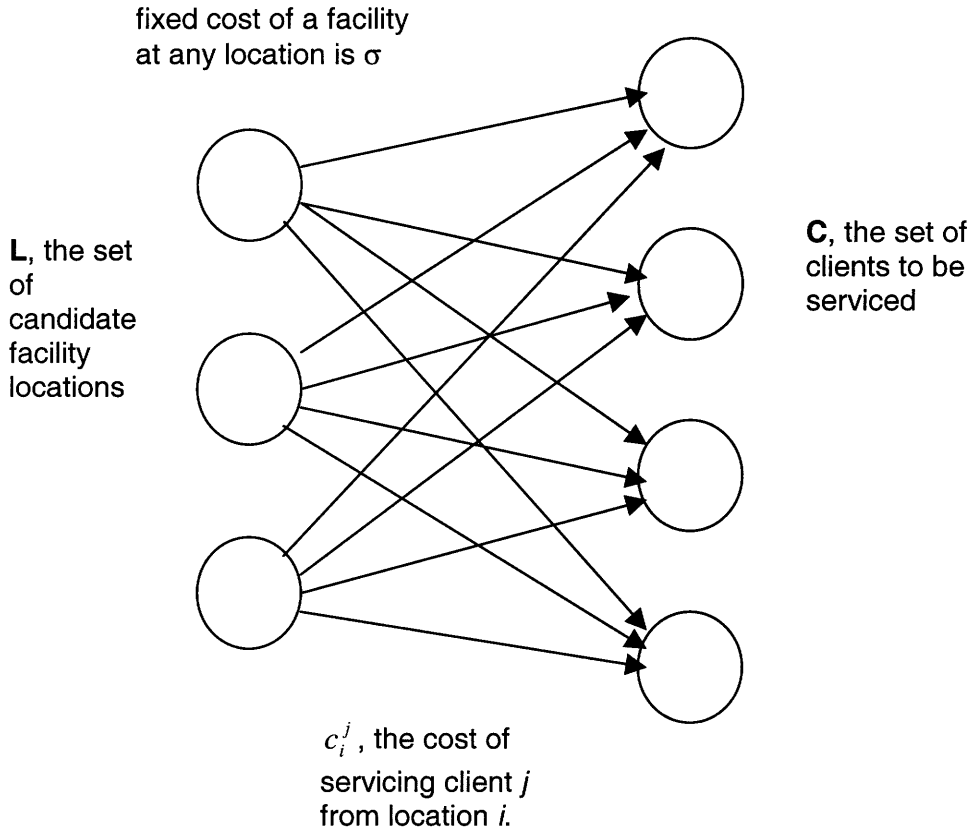

Figure 3. A graph to illustrate UFLP. A subset of locations is chosen to service all clients to minimize total fixed plus servicing costs.

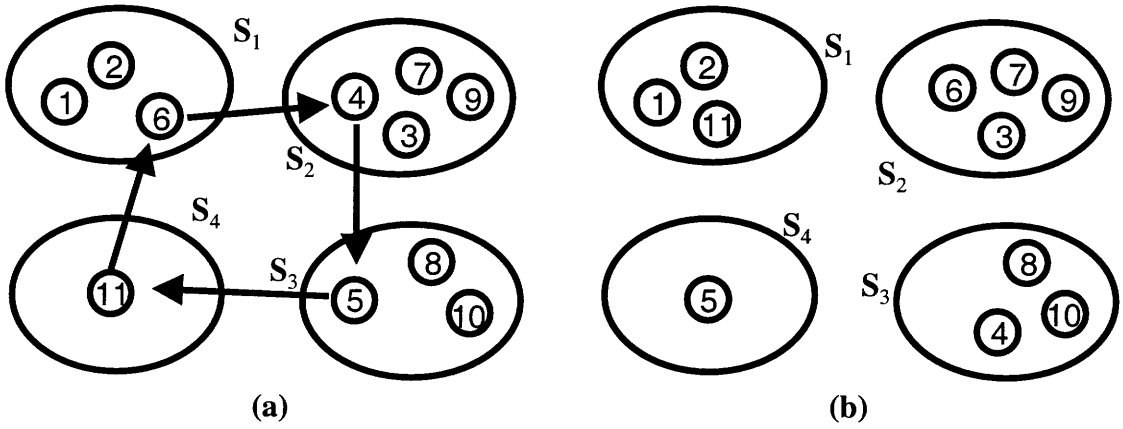

Figure 4: (a) Initial clustering, (b) Clustering after cyclic exchange 6 – 4 – 5 – 11 – 6.

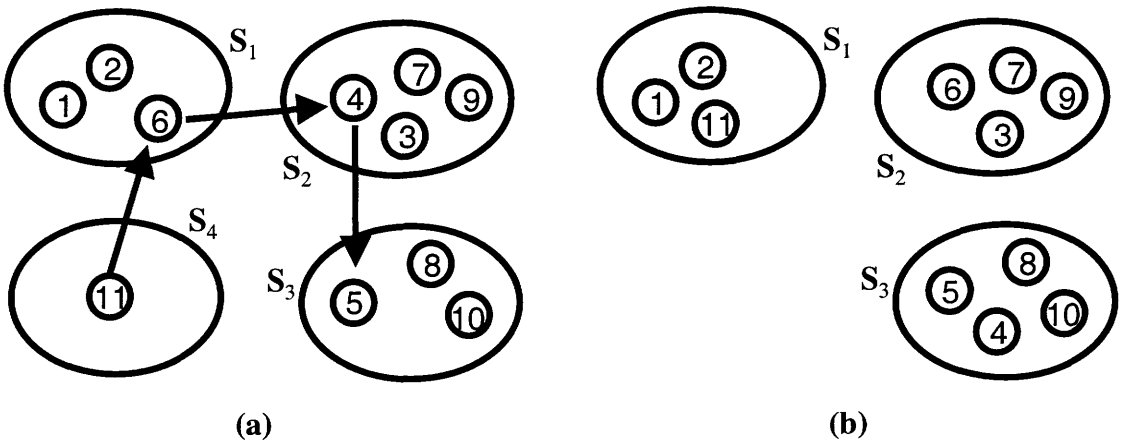

Figure 5: (a) Initial clustering, (b) Clustering after path exchange 11 – 6 – 4 – S_3.

```
FindPathOrCycle( G = (N, A))

begin
    Let P1 = {(i, j)∈A: cij < 0}      # The set of all subset disjoint paths of length 1
    i := 1;
    while size(Pi) > 0 do
    begin
        # Look for subset disjoint paths and cycles in Pi
        for each path P = i - ... - j in Pi do
        begin
            if j is a board type than
            begin
                if Cost(P) + cji < 0 than
                    return cycle i - ... - j - i;
            end
            else return path i - ... - j;
        end
        # Find the path set Pi+1
        for each path P = i - ... - j in Pi do
        begin
          for each edge (j, k) in A such that i - ... - j - k is subset disjoint
              if Cost(P) + cjk < 0 then
                  add path i - ... - j - k to Pi+1;
         end
    end
end.
```

Figure 6: Algorithm for finding negative cost subset-disjoint paths and cycles in a graph.

	Mix	Var.	Prof.	Setup cost	ME Nbd.	Best Clustering	IP Optimal
					(objective value, number of clusters)		
1	32	16	Flat	11000	5577218, 5	5580694, 5	5577218, 5
2	100	100	Flat	31000	42057513, 27	42097933, 28	*
3	10	16	60/40	66200	2683010, 3	2687385, 4	2683010, 3
4	32	16	60/40	6800	673052, 6	675758, 7	673052,6
5	10	16	80/20	154800	2356487, 4	2356487, 4	2356487, 4
6	32	100	80/20	748900	28846510, 8	29414535, 9	28846510, 8
7	100	16	Flat	3000	7241320, 21	7246040, 20	*
8	32	16	Flat	700	814009, 5	814260, 5	814009,5
9	32	16	60/40	69600	6942929, 6	6956230, 7	6942929, 6
10	10	100	60/40	320400	16043774, 5	16043774, 5	16043774, 5
11	100	16	80/20	50000	4178090,14	4281886, 15	*
12	32	16	80/20	14300	456590, 5	458146, 5	456590, 5

Table 1 . Results of Multi-Exchange Neighborhood Search Heuristic for 12 test problems. Alongside each objective function value attained are given the best reported heuristic value from Norman [18], and optimal solutions (where available) as reported by Cohn et al. [17] An asterisk * means that an optimal solution is not known.