Board Scheduling for Circuit Board Assembly: Computational Testing of an Integer Programming Approach

by

Natalie F. Zerangue

Submitted to the
Office of Honors Programs and Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements for

1998-99 UNIVERSITY UNDERGRADUATE RESEARCH FELLOWS PROGRAM

April 15, 1999

Approved as to style and content by:

[illegible]

Wilbert Wilhelm, Faculty Advisor
Department of Industrial Engineering

[illegible]

Susanna Finnell, Executive Director
Honors Programs and Academic Scholarships

Project Organization

(1) Abstract

Circuit boards are integral parts of almost all electronic products, which are widespread in today's society. Companies utilizing circuit board assembly processes must remain competitive by using new technologies, shortening assembly cycle times, and decreasing costs. Optimization techniques involve process planning and lead to shortened cycle times and reduced costs.

No generic workload-balancing tool is available to optimize the assembly process. This study is part of a larger, ATP-sponsored project that will create a tool of this sort by choosing and configuring the placement machines, allocating parts to appropriate machines, assigning parts to machine feeders, arranging part placement in sequential order, and sequentially scheduling boards on particular lines. My part of the project is to assign circuit boards to assembly lines and to sequence the boards on those lines.

The project required that the following factors be considered: the different machine types, the number of machines on each line, the number of different circuit board types, the number of circuit boards of each type to be produced, the number of different component types, and the number of each component type on each circuit board type. Incorporating these considerations, the mathematical model assigns and schedules boards on assembly lines. AMPL, Advanced Mathematical Programming Language, uses script and data files to run with the model. Testing the model and analyzing the results proved that the model is a heuristic that is capable of scheduling printed circuit boards to assembly lines. The ATP-sponsored research project will result in a generic decision support tool for circuit board assembly optimization.

(2) Goal

The goal of this research project is to create a heuristic that will balance workloads for placement machines on printed circuit board assembly lines. The model will efficiently schedule printed circuit boards to the assembly lines to balance machine workload.

(3) Overview

(3.1) Research Team

The research team consisted of Dr. Wilbert E. Wilhelm, Tac Kilavuz, and myself. Dr. Wilhelm is a Professor of Industrial Engineering at Texas A&M University, and Tac Kilavuz is a graduate student in Industrial Engineering at Texas A&M University. The team consulted Robert Fourer, co-creator of AMPL and Professor at Northwestern University, on occasion concerning AMPL issues.

(3.2) Motivation

Circuit boards are vital parts of most electronic products, and circuit board assembly processes exist around the world. Companies must remain competitive by using new technologies, shortening assembly cycle times, and decreasing costs. Planners must frequently optimize the assembly processes to incorporate improvements in technologies and circuit boards.

In addition to assembly technology improvements, electrical components are also changing rapidly. Components are continually becoming smaller and more powerful with reduced distance between leads. These component developments have led and will continue to lead to increasing numbers of components on smaller circuit boards.

Optimizing circuit board assembly lines maximizes throughput and decreases costs. Since purchasing and operating circuit board assembly lines easily cost millions of dollars, optimizing the lines by balancing machine workloads is vital to competitiveness. A scheduling heuristic that balances workloads for machines would increase throughput and reduce costs for any assembly line.

(3.3) Process Description

The printed circuit board assembly process consists of the following basic elements: line, machine, position, board, and component. Each assembly process requires an assembly line, and each assembly line must have placement machines assigned to certain positions on the line. Each machine on the line places certain component types on circuit boards.

Each of the basic elements contains many variations. Some lines may be limited to the assembly of certain board types for technological reasons. Each line may have a different number of positions based on the board types that must be assembled. Many types of machines exist, and each one has different speed capabilities and can place certain component types. In addition to these issues, countless types of boards and components are used, and new ones are frequently designed.

The many steps of printed circuit board assembly must be optimized individually and globally in each assembly process. The planners must consider machine capabilities such as part type versatility and placement speed and accuracy. Machines contain various numbers and sizes of feeders, which feed reeled parts to placement nozzles. For each machine, the planner must determine the sequential placement of parts and the appropriate feeder locations to optimize machine utilization. In addition to using placement machines, the assembly processes may also contain cleaning operations, reflow ovens, and inspection stations, which all contribute to assembly cycle time.

(3.4) Background Information

Previous work has been done in the areas of circuit board assembly process planning, system design, and integer programing. Process planning studies focus on many issues including feeder assignment, placement order, and computer planning tools. Some placement suppliers today provide software with their products, but the software does not deal with placement machines made by other manufacturers. Also, this software and other current process planning methods find only good, not necessarily optimal, solutions to problems (Wilhelm, 1997).

Like the process planning methods, the present system design methods are often not comprehensive and provide only heuristic solutions. Considering cycle time requirements and job sequencing, balancing techniques minimize the number of placement machines. Adding to these considerations to minimize total cost, more comprehensive design models choose the machine and assign the tasks for each workstation (Wilhelm, 1997).

More comprehensive design models greatly increase run time for integer programs and sometimes require run times too long, for practical purposes, to find optimal solutions. Two integer programming research areas are relevant to this project. Branch and Price (B&P) is a column generation technique using specific principles to decompose an integer program. The result is a linear program for the master problem and an integer program for the subproblem, which can be broken down into a discrete number of independent subproblems in some cases. The B&P technique has been used successfully for many scheduling and routing applications (Barnhart, Johnson, Nemhauser, Salvesbergh, & Vance, 1997). In addition to the B&P approach, integer programming research also uses strong cutting plane methods to solve large-scale problems. Cutting plane methods have progressed from solving symmetric traveling-salesman problems involving 150 cities in 1990 to solving similar problems involving over 13,000 cities (Lawler, Lenstra, & Rinnooy Kan, 1998).

(3.5) ATP Project

The ATP research project, funded by the Texas Advanced Technology Program (ATP) and directed by Dr. Wilbert E. Wilhelm, interacts with industry to assist the modeling process, has formed integer programming models to aid in process planning decisions, is evaluating the models for effectiveness and computational accuracy, and will create a generic decision support tool for circuit board assembly optimization. The many aspects of the project include choosing and configuring the placement machines, allocating parts to appropriate machines, assigning parts to feeders, and arranging part placement in sequential order.

No generic workload-balancing tool is available to optimize the assembly process. By evaluating the previously mentioned aspects of the process, the project will result in a decision support tool, which will optimize circuit board assembly by balancing placement machine workloads.

(4) Objectives

The objectives of this research project are the following:

1. Formulate an integer program for the scheduling of boards,
2. Express the integer program as an AMPL mathematical model,
3. Design data and script files for the model, and
4. Computationally test and analyze the model.

(5) Initial Research

At the onset of the project, grouping the circuit boards was a significant part of the scheduling model. Board grouping assigns board types containing common components to the same family, and lines assemble a board type in one family after assembling a different board type in the same family. Assembling boards with common components one after the other reduces the number of components and component types that operators must set up on each machine between assemblies of different board types. The reduction in components and component types decreases the total amount of set up time. Since assembly rates can run thousands of components per hour and machine setup easily takes up to an hour, reducing machine setup time offers a practical way to lower the process cycle time (Maimon & Shtub, 1990).

Another factor to consider when grouping boards is the level of family commonality. If a model requires high commonality, then only the board types sharing the majority of their component types will be members of the same family. A model using low commonality requires less common components between boards and will typically have more board types in each family. Low commonality raises an additional consideration of assigning board types to multiple families. Assigning each board type to one family greatly simplifies but often restricts the model. At the other extreme, allowing multiple assignments of board types may overly complicate the model and leave it too open for error. Also, based on the desired level of family commonality, common components on boards within a particular family, the model can include a parameter that will consider this level and group the boards accordingly. Each level of commonality results in vastly different families and board schedules (Maimon & Shtub, 1990).

Board grouping is beneficial in situations such as those that produce a high variety of board types, each in low volume. The primary interest of this project is a shop environment, in which relatively few board types are produces, each in relatively high volume. Therefore, the heuristic does not incorporate board grouping techniques into the scheduling approach.

(6) Methodology of Objectives

(6.1) Integer program

Building a generic integer programming model that accounts for the requirements and constraints of the circuit board assembly line is the crux of the research project. Using one model to run many experiments allows comparisons of the different tests and true optimization of systems after considering all impacting factors. Each factor will be analyzed individually and together with the other factors and with the cycle time. To reduce program run time in large experiments, the integer program can be relaxed and run as a linear program (LP).

(6.1.1) Preparation

(6.1.1.1) Indices

A working knowledge of the printed circuit board assembly process is vital to the development of a scheduling program. The basic factors of the process are the line, machine type, machine position on a line, board type, and component type. Many types of printed circuit boards exist, and each board type utilizes distinct component types. Assembly lines consisting of various machine types in certain positions must place the component types on the corresponding board types. Shown below, these indices are the basis for all sets, parameters, and variables in the program.

ℓ = line type, $\ell \in L$
m= machine type, $m \in M$
k = machine position on line ℓ, $k \in K_\ell$
b = board type, $b \in B$
c = component type, $c \in C$

(6.1.1.2) Sets

The sets used in the model directly correspond to the indices listed above and are listed below. Five sets define the assembly lines, machines types, machine positions for each assembly line, board types, and component types.

Line :={ℓ in 1..L};
Machines :={m in 1..M};
Position{line} :={k in 1..Kl};
Board :={b in 1..B};
Component :={c in 1..C};

The remaining sets are subsets of the first five sets. Although a factory may have many assembly lines, only certain lines can assemble specific board types. Particular machines types place certain component types. Using the data for machine types and the location of each machine on the line, one set contains the line positions capable of placing particular component types. Within the set of component types is the set of component types on each board, and within this subset is the set of component types on each board that each line and position can place.

LAssembleB_b	= set of lines that can assemble board type b, $\text{LAssembleB}_b \subset$ Line
MachinesC_c	= set of machine types that can place component type c, $\text{MachinesC}_c \subset$ Machines
$\text{PositionCL}_{c\ell}$	= set of positions on line ℓ that can place component type c, $\text{PositionCL}_{c\ell} \subset \text{Position}_\ell$
ComponentsB_b	= set of component types required on board type b, $\text{ComponentsB}_b \subset$ Component
$\text{ComponentsBK}_{b\ell k}$	= set of component types required on board type b that can be placed in position k, $\text{ComponentsBK}_{b\ell k} \subset \text{ComponentsB}_b$

(6.1.1.3) Parameters

In order to effectively schedule the circuit boards, the model must refer to certain known data. In addition to knowing L, M, Kl, B, and C that are the cardinalities of the five basic sets, other parameters are vital to the effectiveness of the model. With multiple machine types and possible combinations of machines in different positions on lines, the model must know the position of a machine to properly assign components. $\text{Mkl}_{\ell k}$ represents the machine type at position k on line ℓ.

Each machine type may place component types differently. All types use nozzles to pick up a component from a machine storage location and place the component on a circuit board. Some machines store special components in trays while other machines store reeled components in feeders. Most machines can hold a different number of component types based on the number of machine feeder slots and the number of slots required by each component type. Small component types fit on a narrow reel and require only one feeder slot, but larger component types need wider reels and may require more than one slot. The number of feeder locations on a machine type and the number of slots required by component types used on that machine both component assignment to that machine. Therefore, the model incorporates the parameter $F_{\ell k}$, which symbolizes the number of feeder locations on a machine of type m, and the parameter $f_{c\ell k}$, which represents the number of feeder slots that component type c requires on the machine in position k on line ℓ.

The cycle time and the available work time are very important factors when minimizing the maximum workload of each line. The cycle time for boards of a particular board type depends on the number of boards of that type, the component types on the board, the number of components of each type, the setup time for that board type, and the placement time for each component type on the machine to which that type is assigned. V_b gives the number of boards of type b to be assembled. The set ComponentsB_b records the component types on board type b, and N_{bc} stands for the number of components of each type on that board type. Although each board type requires a setup time for changeover, the model assumes constant program changeover times, which are not sequence-dependent, and focuses on the setup time for component types on each board type. $S_{c\ell k}$ symbolizes the setup time for component type c on the machine at position k on line ℓ. The parameter $t_{bc\ell k}$ represents the time required to place a component of type c by a machine of type m at position k on line ℓ. The parameter Total_b considers the number of components of each type on board type b and represents the total number of components on that board. Using only the parameters listed above, the model will determine a board's cycle time but will not be able to schedule boards on lines. The available working time of each machine is vital to the scheduling issue, so the parameter $A_{\ell k}$ is the availability, in minutes, of machine type m at position k on line ℓ.

(6.1.1.4) Decision Variables

The model assigns values to the decision variables based on its constraint equations and its parameter values. The constraints will be discussed in detail later. W is the maximum workload on any machine on any line. To know a value for W, the model must also know $W_{k\ell}$, the workload at each position k on every line ℓ. The last two decision variables are binary, and, thus, receive values of either one or zero. $Y_{bc\ell k}$ receives the value of one if component type c on board type b is assigned for placement by machine type m at position k on line ℓ and receives the value of zero if this condition is not true. Similarly, the variable $Z_{b\ell}$ equals one if board type b is assigned to line ℓ and is otherwise equal to zero.

(6.1.2) Formulation

The objective function is to minimize the maximum workload assigned to any machine on any line. The variable W will receive the value of the highest workload.

(6.1.2.1) Minimize W

subject to:

The maximum workload on any line is not greater than the maximum workload at any position on any line.

(6.1.2.2) $\quad W - W_{\ell k} \geq 0 \qquad \ell \in L, k \in K_\ell$

The workload at any position on any line is defined as the component setup time $S_{c\ell k}$ plus the component placement time $t_{bc\ell k} N_{bc} V_b$ summed over all component types $c \in C_{b\ell k}$ and all board types $b \in B$ assigned to position k on line ℓ.

(6.1.2.3) $$W_{\ell k} - \sum_{b \in B} \sum_{c \in C_{b\ell k}} (t_{bc\ell k} N_{bc} V_b + S_{c\ell k}) Y_{bc\ell k} \geq 0$$

$$b \in B, c \in C_{b\ell k}, \ell \in L_b, k \in K_\ell$$

The maximum workload at any position on any line must not be greater than the availability of the machine at that position on that line.

(6.1.2.4) $\quad W_{\ell k} - A_{\ell k} \leq 0 \qquad \ell \in L, k \in K_\ell$

Each board must be assigned to one line. If extremely large quantities of a particular board type must be produced, assigning the board type to more than one line may be a desirable alternative. However, this model requires that large quantities of a particular board type will be assembled on the same line.

(6.1.2.5) $\quad \sum_{\ell \in L} Z_{b\ell} = 1 \qquad b \in B$

The component type c on board type b must be placed by exactly one machine on the line to which board type b is assigned.

(6.1.2.6) $\quad \sum_{k \in K_{c\ell}} Y_{bc\ell k} = Z_{b\ell} \qquad b \in B, c \in C_{b\ell k}, \ell \in L_b$

Assigning component type c on board type b to a machine at position k on line ℓ assumes that board type b is also assigned to line ℓ.

(6.1.2.7) $Y_{bc\ell k} \leq Z_{b\ell}$ $b \in B, c \in C_{b\ell k}, \ell \in L_b, k \in K_\ell$

The feeder mechanism capacity $F_{\ell k}$ limits the number of component types that can be placed by the machine at position k on line ℓ, allowing for variable component widths.

(6.1.2.8) $\sum_{c \in C_{b\ell k}} f_{c\ell k} Y_{bc\ell k} \leq F_{\ell k}$ $b \in B, \ell \in L_b, k \in K_\ell$

As stated earlier, the variables $Y_{bc\ell k}$ and $Z_{b\ell}$ must be binary.

(6.1.2.9) $Y_{bc\ell k} \in \{0,1\}$ $b \in B, c \in C_{b\ell k}, \ell \in L_b, k \in K_\ell$

(6.1.2.10) $Z_{b\ell} \in \{0,1\}$ $b \in B, \ell \in L_b$

The last two constraints in the mathematical model ensure nonzero variable values.

(6.1.2.11) $W \geq 0$

(6.1.2.12) $W_{\ell k} \geq 0$ $\ell \in L_b, k \in K_\ell$

(6.2) AMPL Mathematical Model

The Advanced Mathematical Programming Language, AMPL, is a matrix-generating language that is capable of formulating integer and linear programming problems.

(6.2.1) Parameters

To define a parameter in AMPL, write "param" followed by the parameter name. For example, AMPL defines a simple parameter such as C, the upper bound for the set component, by writing "param C." More complex parameters, such as N_{bc} and $t_{bc\ell k}$, are defined by the following statements:

```
param N {board, component};
param t {board, component, ℓ in line, positionL[ℓ]} >= 0;
```

AMPL does not require the parameter to be set equal to a particular value or a range of values in the model.

(6.2.2) Sets

AMPL defines sets using a similar format. The following sets provide descriptive examples of set definition.

```
set component := {1..C};
set ComponentsB {board} within component default {};
```

The statement "default {}" initializes the set ComponentsB$_b$ to the null set and allows the set elements to change throughout the program.

(6.2.3) Decision Variables

The AMPL format for defining variables is also similar to formats for defining parameters and sets. Y$_{bc\ell k}$ is defined below:

var Y {board, component, ℓ in line, positionL[ℓ]} binary;

The word "binary" at the end of the variable definition restricts the variable to be binary.

(6.2.4) AMPL Formulation

The numbers in this section correspond to the numbers in section (6.1.2). For example, sections (6.2.4.1) and (6.1.2.1) both refer to the objective function, (6.2.4.2) and (6.1.2.2) both refer to the first constraint, and so on.

(6.2.4.1) minimize Maxworkload: W;

(6.2.4.2) subject to MaximumWorkload {ℓ in line, k in positionL[ℓ]}:

W – Wlk[ℓ,k] >= 0;

(6.2.4.3) subject to MachineWorkload {ℓ in line, k in positionL[ℓ]}:

Wlk[ℓ,k] – sum {b in board, c in ComponentsBK[b,ℓ,k]}(((t[b,c,ℓ,k] * N[b,c] * V[b]) + S[c,ℓ,k]) * Y[b,c,ℓ,k]) >= 0;

(6.2.4.4) subject to Availability {ℓ in line, k in positionL[ℓ]}:

A[ℓ,k] - Wlk[ℓ,k] >= 0;

(6.2.4.5) subject to BoardAssignment {b in board}:

sum {ℓ in LAssembleB[b]} Z[b,ℓ] = 1;

(6.2.4.6) subject to OneComponentOneMachine {b in board, ℓ in LAssembleB[b], c in ComponentsB[b]}:

sum {k in positionL[ℓ]} Y[b,c,ℓ,k] - Z[b, ℓ] = 0;

(6.2.4.7) subject to BoardOnLine {b in board, ℓ in LAssembleB[b], k in positionL[ℓ], c in ComponentsBK[b, ℓ,k]}:

Y[b,c,ℓ,k] - Z[b, ℓ] <= 0;

(6.2.4.8) subject to FeederCapacity {b in board, ℓ in LAssembleB[b], k in positionL[ℓ]}:

sum {c in ComponentsBK[b, ℓ,k]} f[c,ℓ,k] * Y[b,c,ℓ,k] <= F[ℓ,k];

The decision variable definitions fulfill the binary constraints, (6.2.4.9)-(6.2.4.10), and the non-zero constraints, (6.2.4.11)-(6.2.4.12). Refer to section (6.2.3) for an example.

(6.3) Experiment Design

AMPL solves the mathematical model with input from data and script files. The data file assigns values to simple parameters, as shown below.

param L := 5;

The script file is much more complex than the data file, using statistical distributions and standard programming formats. Before each random number generation from a distribution, the statement "option randseed" followed by a random five-digit number should be written. This ensures independence between random number streams and creates more realistic results. The number of boards of each board type b, V_b, may range uniformly from a lower bound VL to an upper bound VU. Its script file statement would be the following:

```
let {b in 1..B} V[b]:=round(Uniform(VL,VU));
```

By assuming that every machine has the available workload time of 480 minutes, an eight hour work day, A[ℓ,k] would be defined as it is below.

```
let {ℓ in 1..L, k in 1..Kl} A[ℓ,k]:=480;
```

The first loop in the script file assigns component types to the set ComponentsB[b] from the set of all component types. The assignment is done based on random number generation using the uniform distribution. For each board type b, the loop sees each component type c and randomly decides whether that component type is on that board type. If the component type is on the board type, then component type c becomes an element in the set ComponentsB[b].

```
(6.3.1)
for {b in 1..B, c in 1..C}{
        if Uniform(0,1)>Uniform(0,1) then {
                let ComponentsB[b] := ComponentsB[b] union {c};
        }
};
```

The loop does not ensure the assignment of component types to all board types. Therefore, if the set ComponentsB[b] for board type b is empty, then the next loop assigns a randomly generated component type c to be on board type b. The parameter cnumber represents a component type c, from the set component, that is randomly generated using the uniform distribution .

```
(6.3.2)
for {b in 1..B}{
        if card(ComponentsB[b]) = 0 then {
                let cnumber:=round(Uniform(1,C));
                let ComponentsB[b]:=ComponentsB[b] union {cnumber};
        }
};
```

Since the value of N[b,c] depends on whether component type c is on board type b, it cannot be assigned until after set ComponentsB[b] is complete for each board type. The following loop uses the uniform distribution, ranging from the lower bound of NL to the upper bound of NU, to randomly generate a value for N[b,c] for each component type c on board type b. It also calculates Total[b] by summing N[b,c] values for each board type b.

```
(6.3.3)
for {b in 1..B, c in 1..C}{
        let N[b,c]:=round(Uniform(NL,NU));
        let Total[b]:=Total[b] + N[b,c];
};
```

Since no machine type can place all component types, components must be assigned to certain machine types. Frequently used components can often be placed by many machines, so the loop assumes that the majority of machine types can place the majority of component types. If randomly chosen, machine type m becomes an element in the set MachinesC[c].

```
(6.3.4)
for {c in 1..C, m in 1..M}{
        if Uniform(0,1) > Uniform(0,1) then let MachinesC[c]:=MachinesC[c] union {m};
};
```

Similar to the loop in (3.1.1), this loop does not guarantee that every component can be placed by at least one machine type. Thus, if the set MachinesC[c] for component type c is empty, then the following loop assigns a randomly generated machine type m to the set. The parameter mnumber symbolizes a randomly generated machine type m.

```
(6.3.5)
for {c in 1..C}{
        if card(MachinesC[c]) = 0 then {
                let mnumber:=round(Uniform(1,M));
                let MachinesC[c]:=MachinesC[c] union {mnumber};
        }
};
```

The subsequent loop assigns line types to the set LAssembleB[b] for each board type. It follows the same format as the loop that assigned machine types to the set MachinesC[c]. Line ℓ becomes a member of the set LAssembleB[b] if randomly chosen.

(6.3.6)
```
for {b in 1..B, ℓ in 1..L}{
        if Uniform(0,1) > Uniform(0,1) then let LAssembleB[b]:=LAssembleB[b] union { ℓ };
};
```

Again, this loop does not guarantee that every board can be assembled by at least one line. Therefore, if the set LAssembleB[b] for board type b contains no lines, then the next loop will assign a randomly generated line ℓ to the set. The parameter Lnumber symbolizes a randomly generated machine type ℓ.

(6.3.7)
```
for {b in 1..B}{
        if card(LAssembleB[b]) = 0 then {
                let Lnumber:=round(Uniform(1,L));
                let LAssembleB[b]:= LAssembleB[b] union {Lnumber};
        }
};
```

Since the model now knows the machine types that can place certain component types, MachinesC[c], the next step is to use the known information to logically determine the set positionCL[c, ℓ], the set of positions on line ℓ that can place component type c. Every machine type m, regardless of its position on a line, can place the same component types. If Mkl[ℓ,k] is in MachinesC[c], then the machine type at position k on line ℓ is one of the machine types that can place component type c. If this is true, then position k is an element in the set positionCL[c, ℓ], meaning that position k is one of the positions on line ℓ that can place component type c.

At least one position on line ℓ must be capable of placing each component type. Thus, if component type c could not originally be placed at any position k on line ℓ, leaving the set positionCL[c, ℓ] empty, then a randomly chosen position k becomes a member of the set positionCL[c, ℓ]. Thus, each line ℓ can place every component type c, but not every line ℓ can assemble board type b (refer to section (6.3.6)), which may require component type c.

```
(6.3.8)
for {c in 1..C, ℓ in 1..L}{
        for {k in 1..Kl}{
                if Mkl[ℓ,k] in MachinesC[c] then
                        let positionCL[c,ℓ]:=positionCL[c,ℓ] union {k};
        }
        if card(positionCL[c,ℓ]) = 0 then {
                let Pk := round(Uniform(1,Kl));
                let positionCL[c,ℓ] := positionCL[c,ℓ] union {Pk};
        }
};
```

Knowing the elements in the sets ComponentsB[b] and positionCL[c, ℓ], the model assigns the appropriate members to the set ComponentsB[b]. Basically, since board type b requires certain component types and each component type can be placed at specific positions on line ℓ, the model combines the information to list the component types that are required on board type b and can be placed at position k on line ℓ.

```
(6.3.9)
for {b in 1..B, c in ComponentsB[b], ℓ in LAssembleB[b], k in positionCL[c,ℓ]}{
        let ComponentsBK[b,ℓ,k]:=ComponentsBK[b,ℓ,k] union {c};
};
```

The last loop in the script file verifies that each component on board type b can be placed by at least one machine on every line on which board type b can be assembled. Circuit board assembly lines usually have two or three component placement machines with different capabilities. Although many machines can place some of the same component types, each machine type has unique feeder capacities and speed capabilities for placing these component types. Most component types are stored on reels, and the reels vary in width. One feeder slot on a machine can hold one reel of a narrow component. However, some reels are wider than one feeder slot and may require two or three slots.

The loop assigns feeder-slot widths to component types by randomly choosing the feeder slot width required by component type c. Based on the component types being used in the scheduling model, the user must assign probabilities that f[c, ℓ ,k], the feeder slot width required by component type c at position k on line ℓ ., equals the values of one, two, or three. The parameter probone is the probablility that f[c, ℓ ,k] equals one, and the difference of (probtwo – probone) is the probablility that f[c, ℓ ,k] equals two. The difference of (1 – probtwo) is the probability that f[c, ℓ ,k] equals three. The parameter fnumber signifies a randomly generated probability and is compared to the probabilities probone and probtwo to determine f[c, ℓ ,k] for component type c. At the end of each position loop, if no positions on line ℓ can place component type c on board type b, then a randomly chosen position k, parameter Pk, is assigned to the set ComponentsBK[b, ℓ ,Pk] for this purpose. At this point, the value of f[c, ℓ ,Pk] is determined as described above.

```
(6.3.10)
for {b in 1..B, ℓ in LAssembleB[b], c in ComponentsB[b]}{
        let counter:=0;
        for {k in positionCL[c, ℓ ]}{
                let counter := counter + 1;
                if f[c, ℓ ,k] = 0 then {
                        let fnumber:=Uniform(0,1);
                        if fnumber < probone then let f[c, ℓ ,k]:=1;
                        else{if fnumber >= probone and fnumber < probtwo then let f[c, ℓ ,k]:=2;
                                else let f[c, ℓ ,k]:=3;
                        }
                }
        }
        if counter = 0 then {
                let Pk := round(Uniform(1,Kl));
                let ComponentsBK[b, ℓ ,Pk] := ComponentsBK[b, ℓ ,Pk] union {c};
                if f[c, ℓ ,Pk] = 0 then {
                        let fnumber:=Uniform(0,1);
                        if fnumber < probone then let f[c, ℓ ,Pk]:=1;
                        else{if fnumber >= probone and fnumber < probtwo then let f[c, ℓ ,Pk]:=2;
                                else let f[c, ℓ ,Pk]:=3;
                        }
                }
        }
};
```

(6.4) Testing and Analysis

Testing of the model is necessary to build an accurate optimization model of the circuit board assembly process. The tests detect both the advantages and the disadvantages of the design, leading to more accurate designs. Both small scale test cases and larger, more realistic test cases, using data from industry, should be performed.

In order to generate solutions to the circuit board assembly optimization problems, a matrix generator must convert the designs into integer and linear programs. These programs can be solved using existing software and additional algorithms interfaced with the software. The solutions from the integer programs will be compared with the solutions from the linear programs. Sometimes, heuristic solutions are the only feasible solutions, but the solutions will be evaluated for attainability and run time requirements.

The model designs and their corresponding solutions should always be organized and analyzed. Organizing the designs into a table format enhances understanding of the designs and allows comparisons to be made. Cycle time is defined as the time to assemble all boards of a particular type, and the analysis considerations include cycle time as a function of number of component types, board types, machine types, and lines. Other contributing design factors are component placement times and setup times and the number of components of each type. The conclusions are drawn based on the analyzed results of these considerations.

(6.4.1) First Test Case

The first test case was a very simple one that kept all parameters constant except the five basic factors, which are the number of component types, board types, lines, positions on each line, and machine types. The level for each factor was intentionally kept at two to keep the problem size small. The assigned parameter values are listed below.

```
param C:=2;             # cardinality for set of component types
param B:=2;             # cardinality for set of board types
param L:=2;             # cardinality for set of lines
param Kl:=2;            # cardinality for set of positions
param M:=2;             # cardinality for set of machine types

param VL:=5;            # lower bound of V[b]
param VU:=5;            # upper bound of V[b]
param NL:=10;           # lower bound of N[b.c]
param NU:=10;           # upper bound of N[b,c]
param tL:=.001;         # lower bound of t[b,c,ℓ,k]
param tU:=.001;         # upper bound of t[b,c,ℓ,k]
param SL:=3;            # lower bound of S[c,ℓ,k]
param SU:=3;            # upper bound of S[c,ℓ,k]
param probone:=.7;      # probability that f[c,ℓ,k]=1
param probtwo:=.9;      # probability – probone that f[c,ℓ,k]=2
```

After running the model and script file in AMPL, the sets contained the following values. Realistically, the set ComponentsBK[b,c,ℓ,k] can be an empty, meaning that the machine type at position k on line ℓ cannot place component type c on board type b.

```
set LAssembleB[1] := 2;
set LAssembleB[2] := 1;
set MachinesC[1] := 1;
set MachinesC[2] := 2;
set positionCL[1,1] := 2;
set positionCL[1,2] := 2;
set positionCL[2,1] := 1 2;
set positionCL[2,2] := 1;
set ComponentsB[1] := 1;
set ComponentsB[2] := 2;
set ComponentsBK[1,1,2] := 1;
set ComponentsBK[1,2,2] := 1;
set ComponentsBK[2,1,1] := 2;
set ComponentsBK[2,1,2] := 2;
set ComponentsBK[2,2,1] := 2;
set ComponentsBK[1,1,1] := ;        # empty
set ComponentsBK[1,2,1] := ;        # empty
set ComponentsBK[2,2,2] := ;        # empty
```

When AMPL runs the program, it creates three files, a *.col file, a *.row file, and an *.mps file. Respectively, these define the columns, rows, and created coefficient values belonging to the columns and rows. Many different mathematical programming solvers will solve this problem. Using the *.mps file, the CPLEX solver output includes the following data. The word "objective" signifies W, the objective function, which has a value of 3.05 minutes. The small objective function value points out the simplicity of this example.

```
MIP Presolve eliminated 15 rows and 7 columns.
Aggregator did 1 substitutions.
Reduced MIP has 4 rows, 4 columns, and 8 nonzeros.

Integer optimal solution:  Objective =  3.0500000000e+000
Solution time =   0.47 sec.   Iterations = 3   Nodes = 0
```

The column variable values increase the clarity of the solution. These are listed below. All variables that are not listed have zero values.

```
Wlk[1,1]      := 3.05
Wlk[1,1]      := 3.05
W             := 3.05
Y[1,1,2,2]    := 1
Y[2,2,1,1]    := 1
Z[1,2]        := 1
```

```
Z[2,1]        := 1
```

By charting the matrix on a spreadsheet, the interactions between variables in the columns and rows become much more clear. Refer to Appendix One to view the spreadsheet.

(6.4.2) Second Test Case

The second test case keeps the parameters for the five basic factors constant at a value of two, as in section (6.4.1) and sets the other parameters equal to more realistic values. The different values are listed below.

```
param VL:=5;        # lower bound of V[b]
param VU:=50;       # upper bound of V[b]
param NL:=1;        # lower bound of N[b.c]
param NU:=50;       # upper bound of N[b,c]
param tL:=.0001;    # lower bound of t[b,c,ℓ,k]
param tU:=.1;       # upper bound of t[b,c,ℓ,k]
param SL:=1;        # lower bound of S[c,ℓ,k]
param SU:=5;        # upper bound of S[c,ℓ,k]
```

The output for this example included the following information.

```
MIP Presolve eliminated 15 rows and 7 columns.
Reduced MIP has 5 rows, 5 columns, and 10 nonzeros.

Integer optimal solution:  Objective =   3.4697116250e+001
Solution time =    0.47 sec.   Iterations = 6   Nodes = 2
```

The objective function has a value of about 35 minutes. Adding variability into the process by increasing the range of certain parameters significantly increased the maximum workload for this small scale test case.

(6.4.3) Third Test Case

By leaving the lower and upper bound parameters as they were in section (6.4.1) and changing the levels of the five factors, the program produces different results. These values are listed below.

Appendix One: Matrix spreadsheet for section (6.4.1)

	Wlk [1,1]	Wlk [1,2]	Wlk [2,1]	Wlk [2,2]	W	Y[1,1,1,1]	Y[1,1,2,2]	Y[2,2,1,1]	Y[2,2,2,2]	Z[1,2]	Z[2,1]	Z[2,2]
MaximumWorkload[1,1]	-1				1							
MaximumWorkload[1,2]		-1			1							
MaximumWorkload[2,1]			-1		1							
MaximumWorkload[2,2]				-1	1							
MachineWorkload[1,1]	1					-3.05		-3.05				
MachineWorkload[1,2]		1										
MachineWorkload[2,1]			1									
MachineWorkload[2,2]				1			-3.05		-3.05			
Availability[1,1]	-1											
Availability[1,2]		-1										
Availability[2,1]			-1									
Availability[2,2]				-1								
BoardAssignment[1]										1		
BoardAssignment[2]											1	1
OneComponentOne Machine[1,2,1]							1			-1		
OneComponentOne Machine[2,1,2]								1			-1	
FeederCapacity[1,2,1]												
FeederCapacity[1,2,2]							2					
FeederCapacity[2,1,1]								1				
FeederCapacity[2,1,2]												
FeederCapacity[2,2,1]												
FeederCapacity[2,2,2]									2			
Maxworkload					1							

```
param C:=20;        # number of component types
param B:=5;         # number of board type
param L:=3;         # number of lines
param Kl:=3;        # number of positions
param M:=5;         # number of machine types
```

The sets for this test are much larger than in the previous tests. For example, since C=20 and L=3, the number of positionCL[c, ℓ] sets is 60, compared to 4 in section (6.4.1). Also, the number of members in each set increases drastically as the five basic parameters increase. For example, the number of members in the set ComponentsB[b] is much higher since this test case has a higher number of component types. The solution for this example is below.

```
MIP Presolve eliminated 50 rows and 107 columns.
Aggregator did 38 substitutions.
Reduced MIP has 79 rows, 183 columns, and 430 nonzeros.

Integer optimal solution:  Objective =   2.7450000000e+001
Solution time =    0.59 sec.    Iterations = 69   Nodes = 5
```

The objective function W has a value of 27.45, which is between the objective function values for test cases one and two. This result seems to signify that variability in the statistical distributions of parameters has a greater impact on maximum workload than simply altering the values of the five basic parameters.

(3.4.1) Fourth Test Case

In the fourth test case, the program operates with variability in the statistical distributions and with increases in the five basic parameters. Although the values for the basic parameters of component type, board type, line, line position, and machine type are smaller than those in section (6.4.3), this example is much more realistic since it involved variation in all aspects of the program. The five basic parameters are listed below, and all other parameters are identical to those in section (6.4.2).

```
param C:=10;        # number of component types
param B:=5;          # number of board type
param L:=3;          # number of lines
param Kl:=3;         # number of positions
param M:=5;          # number of machine types
```

The CPLEX solution is recorded below.

```
MIP Presolve eliminated 59 rows and 79 columns.
Aggregator did 4 substitutions.
Reduced MIP has 47 rows, 71 columns, and 171 nonzeros.

Integer optimal solution:  Objective =  4.0948723412e+002
Solution time =   0.41 sec.   Iterations = 15   Nodes = 1
```

The objective function W is noticeably larger than it is in any of the other test cases. At a value of approximately 409 minutes, the value of W is moderately close to the workload availability of 480 minutes. Since this test case is a larger example than that in section (6.4.1) with the reasonably realistic statistical distributions used in section (6.4.2), the large W value in this example signifies that machine and line workloads increase at a very high rate when combined with distribution variation and slight increases in basic parameters. However, in an industrial situation, as the number of component types and board types increase, the number of line positions and machine types sometime increase while the number of lines must increase. These increases in all basic parameters result in *constraint stabilization* and feasibility of each realistic test case.

(3.4.2) Fifth Test Case

Attempting to solve larger test cases using the same upper and lower parameter bounds that are in section (6.4.2) demonstrated that the bounds for the parameter t[b,c,ℓ,k] are unrealistic. The component placement times created infeasible solutions because all board types could not be assembled within the maximum working time of 480 minutes. As a result, these parameters were changed to the following:

```
param tL:=.001;
param tU:=.01;
```

The cardinality for the set of component types has increased, but the cardinalities for the other basic sets remained the same.

```
param C:=40;
param B:=5;
param L:=3;
param Kl:=3;
param M:=5;
```

Although the number of component types has increased, the objective function value is only 134 minutes, which is much lower than in section (6.4.4), due to the lower t[b,c,ℓ,k] bounds. The CPLEX output for this example included the information shown below.

```
MIP Presolve eliminated 64 rows and 196 columns.
Aggregator did 78 substitutions.
Reduced MIP has 125 rows, 275 columns, and 668 nonzeros.

Integer optimal solution: Objective =  1.3355022439e+002
Solution time =   1.12 sec.   Iterations = 538   Nodes = 112
```

(3.4.3) Sixth Test Case

The only difference between this test case and the one in section (6.4.5) is the cardinality of the set of component types, as shown below.

```
param C:=100;
```

The AMPL formulation time and the CPLEX solving time were both significantly longer with this test than they were with the other tests. The CPLEX solution shown below was obtained before the best solution for the problem had been found.

```
MIP Presolve eliminated 99 rows and 512 columns.
Aggregator did 232 substitutions.
Reduced MIP has 234 rows, 570 columns, and 1360 nonzeros.

Aborted, integer feasible: Objective =  2.9686441197e+002
Solution time = 1825.24 sec.   Iterations = 805900   Nodes = 175230
```

(4) Conclusion

The goal, creating a heuristic that balances machine workloads on printed circuit board assembly lines, has been reached. The scheduling method is a heuristic because it does not incorporate all of the complex aspects of the circuit board assembly process, producing a balanced workload that also takes these aspects into consideration. At this point in its development, the program oversimplifies some complex constraints, and its results need to be compared to actual circuit board assembly workloads for the same tests. Still, many of the important constraints for the process are part of the model and allow the model to produce feasible solutions.

Some combinations of parameter values do not produce feasible solutions. The extensive testing of the model seems to show that infeasible solutions are realistic problems in the circuit board assembly industry. Not all combinations of component types, board types, lines, positions, and machine types can lead to feasible production schedules for designated time periods. As researchers study this subject further and the model continues to undergo improvements, the results will approach optimality.

(8) Future Considerations

This project is open to future research and development in many areas. In addition to the many companies that produce these machines, which are quite different from each other, each company may have several types of placement machine and distinct capabilities for each model. The diverse considerations for placement machine optimization include the number of nozzle heads placing parts at one time, the size of the nozzle head, the amount of time between each part placement, the delicacy and shape of the parts, the diversity of one part placement to another, and part placement accuracy requirements. Many lines have more than one placement machine and could include multiple machines of the same type or more than one type of placement machine. For example, some placement machines specialize in quickly placing commonly used parts from tape reels while other machines specialize in placing larger, less common parts from both tape reels and trays. Also, machine failure and the rescheduling of boards on different lines are possible occurrences. These factors are both important and complex to model but should be considered.

As research on the scheduling model continues, it will seek to overcome limitations and to consider additional aspects of the circuit board assembly process. The ATP-sponsored project will then use the scheduling model as part of a generic workload-balancing tool, which seeks to optimize the assembly process.