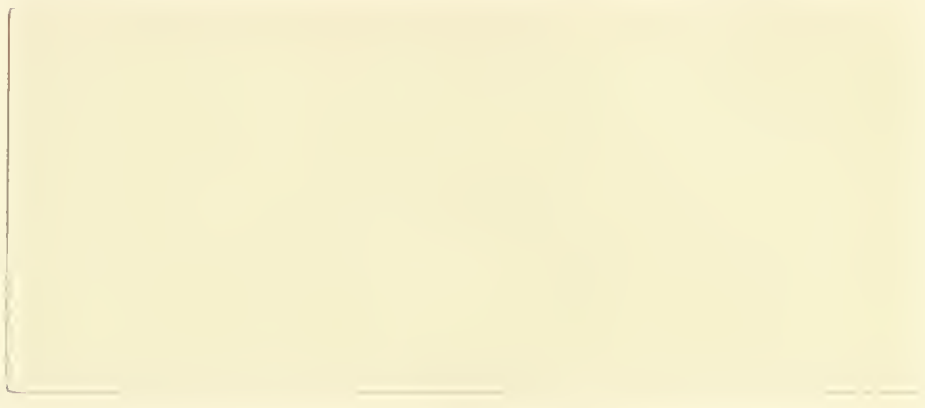
WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Setup Optimization and Workload Balancing for Mixed-model Electronics Assembly Operations

Anantaram Balakrishnan
and
François Vanderbeck

WP# 3551-93-MSA April, 1993

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



Setup Optimization and Workload Balancing for Mixed-model Electronics Assembly Operations

Anantaram Balakrishnan
and
François Vanderbeck

WP# 3551-93-MSA April, 1993

Setup Optimization and Workload Balancing for Mixed-model Electronics Assembly Operations[†]

Anantaram Balakrishnan

Sloan School of Management
M. I. T., Cambridge, MA

François Vanderbeck[*]

Center for Operations Research and Econometrics
Université Catholique de Louvain
Louvain-la-Neuve, Belgium

April 1993

[†] Supported in part by MIT's *Leaders for Manufacturing* program

[*] Partially supported by the Collège Interuniversitaire d'études doctorales dans les sciences du Management (CIM), Belgium

Abstract

To cope with the rapid technological advances, product proliferation, shorter product lifecycles, and increasing competition, electronics companies are adopting flexible assembly strategies using multiple surface mount assembly modules to produce many different products in small batches. The component placement machines account for over half the investment in surface mount assembly lines, and also require considerable setup time (comparable to the actual processing time for small lot sizes) to changeover production from one product to another. This paper addresses a tactical planning problem for high mix, low volume electronics assembly facilities containing several placement machines (modules) operating in parallel. We propose an optimization model to decide which product families to assemble on each machine in order to minimize the total setup cost per demand period while ensuring that none of the machines is overloaded. Unlike strategic and operational models that emphasize either workload balancing or setup optimization, our tactical planning model simultaneously considers both these conflicting factors. To capture the impact of product grouping decisions on setup cost, we consider a partial setup policy that is both easy to implement in practice, and convenient to incorporate in the tactical planning model. The policy consists of reserving certain slots on each placement machine for permanent components, and loading the remaining components as needed to assemble each product.

We formulate the tactical planning problem as an integer program, and show that even the special case of minimizing the setup cost on a single machine for a given assignment of products is NP-hard. For the general model, we develop an optimization-based method, combining column generation, heuristics, and lower bounding procedures, to approximately solve the problem and assess the quality of the solution. As part of our development, we describe solution methods for two practical subproblems–a single machine, product selection subproblem, and the setup optimization subproblem–that might apply directly to short-term production planning. Our computational experience with medium-sized test problems provides insights regarding the effective implementation of column generation strategies in this problem context, and identifies some opportunities for improvement.

Keywords: Electronics assembly, tactical planning, column generation

1. Introduction

Rapid advances in electronic product and process technologies, proliferation of products, shorter product lifecycles, and competitive pressures to reduce cost, lot sizes, and lead time have prompted many electronics companies to adopt flexible assembly strategies using multiple assembly modules that can each assemble many products. With the advent of surface mount and other advanced interconnect technologies, the level of assembly automation has increased considerably, requiring sophisticated equipment for high speed and precision component placement. The investment for a single surface mount assembly line can total several millions of dollars, with placement machines (with associated tooling) alone costing over a million dollars each. On the other hand, as companies strive towards "lot size of one" production to better meet the customers' needs, asset utilization decreases due to more setups. To cope with these trends and maintain their competitive position, electronics companies require principled planning methods and operating practices that ensure efficient and effective use of their manufacturing resources.

This paper addresses a medium-term, tactical decision facing high mix, low volume electronics assembly facilities containing several surface mount assembly modules (one or more placement machines in series, see Section 2.1) operating in parallel, possibly at different sites. We consider facilities in which each module can assemble several different products (with appropriate setups), and modules have overlapping capabilities, i.e., for each product, we have a choice of modules to assemble that product. Furthermore, the demand for individual products is not adequate to justify using dedicated, single-product modules. Operations with these characteristics, which we refer to as mixed-model assembly operations, are common in companies producing electronic subsystems for applications such as telecommunications, computer peripherals, consumer electronics, and medical systems. We will focus on the component placement operation since the automatic placement machines are very expensive and also require considerable setup time. For convenience, we use the words machine, module, and line interchangeably. Given the equipment configuration and capacity of each machine, and the projected demand for various products, the tactical plan determines which products to assemble on each machine.

In making this tactical decision, managers must address two conflicting objectives: reducing the setup or changeover times, and distributing the workload evenly across the parallel machines (relative to their available capacities). When lot sizes are small (say, tens or hundreds of boards per lot), equipment changeover times can equal or exceed the actual processing (placement) times. For instance, the time to change over production from one board type to another on a placement machine can vary from 15 minutes to over an hour

depending on the number of new components to be loaded, whereas placing all the components for a batch of 50 boards with, say, 200 placements per board, might require less than an hour of processing time. Reducing setup times is, therefore, an important concern both to increase productive capacity, and to provide quick response with minimal inventories. On the other hand, balancing the workload has several benefits including reducing overtime expenses, decreasing the lead time to supply board sets (i.e., multiple board types that are assembled into a single unit) for final assembly, and providing safety capacity on each machine to accommodate contingencies. These two objectives can be contradictory. To minimize total setup time, we might assign products that share many common components to the same machine; however, this strategy of grouping products solely based on component commonality can result in imbalances due to uneven group sizes.

To address this tradeoff, we develop an optimization model that assigns products to parallel placement machines (or modules) in order to minimize the total setup cost while ensuring that the processing workload on each machine does not exceed a prespecified limit. Most previous models either emphasize equal workload allocation but capture the setup implications only indirectly, or optimize the setup time for each line assuming a predetermined assignment of products to lines. In contrast, our model provides a structured way to simultaneously consider setup and workload balancing issues. We propose a simple setup policy that is both practical, and enables us to decouple the medium-term product assignment decisions from the detailed (weekly or daily) sequencing and scheduling decisions. The policy, which we call the partial setup policy, consists of reserving certain slots on each placement machine for permanent components, and loading the remaining components as needed to assemble each product. By treating the workload balancing requirements as constraints while optimizing setups, our model provides the capability to generate alternative scenarios with different setup–workload characteristics that managers can evaluate before deciding the best tactical plan for their specific operation.

We formulate the tactical planning problem as an integer program, and develop an optimization-based methodology, combining column generation, heuristics, and lower bounding procedures, to approximately solve the problem and assess the quality of the solution. We can embed this method in a branch-and-bound scheme to solve the problem optimally. As part of our algorithmic development, we describe solution methods for two interesting subproblems–a profit maximizing product selection problem, and a setup minimization subproblem–that can apply directly to support short-term planning and scheduling decisions. Our computational experience using medium-sized problems containing upto 20 products, 60 component types, and 4 machines provides insights

regarding the effective implementation of column generation strategies for this problem context, and identifies some opportunities for improvement.

The rest of this paper is organized as follows. Section 2 motivates and describes our tactical planning model. We briefly review the process flow in electronics assembly operations, justify the structure of our model, discuss its scope and assumptions, and introduce the problem formulation. We also discuss model variants and special cases. Section 3 first provides an overview of the solution methodology before describing its various components, including heuristics, lower bounding techniques, and optimization procedures for the two subproblems. Section 4 reports computational results for our test problems, discussing implementation variants and the performance impact of different algorithmic options and parameters. Section 5 offers concluding remarks.

2. Tactical Planning for Mixed-model Electronics Assembly

We first present an overview of electronics assembly operations, with particular emphasis on component placement, and briefly describe the hierarchy of planning and scheduling decisions for electronics assembly operations. Section 2.2 defines the scope of our tactical planning model, motivates the problem and our modeling approach, and justifies the assumptions. We discuss the modeling challenges in capturing the impact of product assignment decisions on setup times, and describe the partial setup policy that our model assumes. Section 2.3 presents an integer programming formulation for the tactical planning problem, and discusses model variants and special cases.

2.1 Overview of Electronics Assembly Operations

2.1.1 Process flow and component placement operations

Electronics assembly refers to the process of populating bare printed circuit boards with electronic and mechanical components. This paper focuses on the assembly of surface mount components; most new products use surface mount technology (SMT) whenever possible (instead of the older through-hole technology) due to its numerous benefits including smaller board size, lower weight, reduction in electrical noice, improved shock and vibration resistance, and lower board fabrication costs. Broadly, surface mount assembly consists of five major steps–solder paste application, high-speed and precision component placement, soldering, cleaning, and testing. Prasad [1989] describes the process flows for three types of surface mounting, and discusses each assembly operation in detail.

We define a placement "module" as a single component placement machine or several placement machines connected in series by conveyors; a module might also contain dedicated stations to perform other operations such as screen printing, manual assembly, reflow soldering and so on. Thus, a complete assembly line corresponds to a module with dedicated stations for all the surface mount assembly operations from screen printing to testing. Mixed-model assembly facilities employ placement modules in a variety of configurations ranging from a purely functional layout (containing parallel placement modules as one stage, parallel reflow ovens as the next stage, and so on) to mini-lines or complete assembly lines. For expositional ease, we assume that each module has a single placement machine, and so we refer to machines instead of modules or lines. However, as we note later, our model can also incorporate multiple placement machines in series.

The automatic placement machines are the most expensive (accounting for over 50% of the investment in a typical SMT line) and often require the highest changeover time relative to other SMT operations. The numerically-controlled sequential or sequential/simultaneous placement machines (see Prasad [1989]) used in mixed-model assembly facilities have three main components: (i) a table (stationary or movable) to hold the the board (or a panel of boards), (ii) feeder input positions or **slots** on the periphery of the table to hold component feeders, each containing a reel[1] of one component type, and (iii) one or more component placement heads. Each printed circuit board has a bill of components specifying the types of components it requires, the number of components of each type, and the placement location on the board for each component. To assemble a particular board type or product, the operator must first ensure that the feeders carrying each of the required component types are mounted in appropriate slots on the machine. The placement head sequentially picks a component from a feeder position, inspects (optional) and orients the component, and places it at the appropriate location on the board. These "pick-and-place" operations are governed by a process plan in the machine's controller which specifies the component type, X-Y coordinates, and orientation for each placement location on the board, the location of the feeder containing each component type, and the sequence of operations. For tactical planning, we use two parameters–slot capacity and placement speed or placement time per board–to characterize each machine.

The **slot capacity** (sometimes called the component staging capacity) of a machine is the maximum number of feeders that it can hold. We can represent this capacity either in terms of "standard" slots (e.g., number of 8mm tape feeder input slots) or in terms of the

[1] SMT components are dispensed in a variety of formats, e.g., tape and reel, tubes, trays. For simplicity, we refer to all the component presentation and handling formats as reels.

total width of feeders that can be loaded at one time; for convenience, we will assume the first representation.

We refer to the total time to place all the components on a particular board (including the initial board setup time to locate the fiducials, etc.) as the **placement time** or **cycle time** for that board. This placement time is a function of numerous machine-specific and product-dependent parameters including the board area, number of components to be placed on the board, size of the components, relative location of the component feeders, number and speed of the placement heads (and feeder movements, X-Y table), component pickup mechanism and placement head configuration, intermediate operations (e.g., inspection, pin alignment), and placement sequence. Ball and Magazine [1989], Bard, Clayton and Feo [1989] and Gavish and Seidmann [1987] develop cycle time models for different placement machines. We assume that, for each available machine, the planner can incorporate these factors to estimate the average placement speed for each component type or the average placement time per board for every product. We will use these placement time estimates to quantify each machine's workload.

By changing the component feeders and the process plan, the same machine can assemble a wide variety of products. A machine's slot capacity determines its setup requirements. If the machine has a large slot capacity relative to the number of component types required for each product, then we can reduce changeover time by assigning dedicated slots to certain common component types, and keeping these reels on the machine at all times. The setup policy that we describe in Section 2.2 exploits this observation.

2.1.2 Planning hierarchy

The growing emphasis on reducing setups, optimizing cycle time, eliminating work-in-process inventories, and improving asset utilization has motivated considerable management science research in recent years on models to support short-term production planning decisions. The vast majority of this literature deals with optimizing feeder-to-slot assignment and placement sequencing decisions to reduce the cycle time for a single product (see, for instance, Ahmadi, Grotzinger and Johnson [1988], Ball and Magazine [1988], Bard et al. [1989], Drezner and Nof [1984], Francis et al. [1989], and Gavish and Seidmann [1987]). These optimization models tend to be very machine-specific, incorporating features that contribute most to the placement time on each machine (e.g., single versus dual delivery of components, Euclidean versus Manhattan distance metrics to compute head travel time, etc.). At a higher level in the planning hierarchy, researchers have studied short-term board sequencing problems for a single machine or line, and setup strategies to reduce changeover

time to produce a given set of products (e.g., Lofgren and McGinnis [1986], Carmon, Maimon and Dar-El [1989], and Daskin, Maimon and Shtub [1991]).

Some recent literature has begun to address more strategic issues in PCB assembly using models that consider both multiple products and multiple assembly lines. Ahmadi and Matsuo [1992] address equipment configuration decisions in mixed-model assembly facilities using mini-lines. A mini-line is a series of assembly stages, with each stage (e.g., placement, soldering, etc.) containing multiple parallel machines. Ahmadi and Matsuo propose a hierarchical approach to decide the number of mini-lines, group the product families, and allocate machines to each mini-line. Ahmadi and Kouvelis [1992] describe a mathematical programming formulation to represent alternative design approaches–flexible flow lines, mini-lines, and hybrid lines–for electronics assembly. These models minimize makespan (i.e., the maximum processing time over all the lines) to balance the workload, but do not explicitly incorporate the setup interactions across product families since they are concerned mainly with long-term configuration decisions.

2.2 Tactical Planning: Problem Definition and Modeling Issues

2.2.1 Scope of model

We consider an intermediate level of decision-making that incorporates the workload balancing feature of the strategic decision models on the one hand, but also considers a finer grain representation of individual placement machines to capture the setup costs in a high mix, low volume environment. For medium-term planning, we assume that the equipment configuration is fixed, i.e., we know the number of parallel placement machines, and their respective capabilities. We are given the **bill of components** for each product, and its **processing time** (see Section 2.2.3) on every machine. By "product" we mean a single board type or a family of similar board types (e.g., mother boards with nearly identical bill of components)[2]. Unlike short-term production planning models that require detailed information regarding the size, release date, and due date for each batch of products, medium-term planning must rely on aggregate forecasts of production requirements. We use two parameters to represent the production requirement for each product: (i) its projected **demand** in terms of total number of boards required per period[3], and (ii) its anticipated

[2] Double-sided boards require two passes through the assembly line, and so we consider them as two separate products. If necessary, we can specify that both sides must be assembled on the same or different machines.

[3] The demand period (or time unit) might be considerably shorter than the planning horizon; typically, the period is long enough to ensure that every product is assembled at least once per period.

production frequency[4] or the average number of batches assembled during this period. The **tactical planning problem** involves deciding **which products to assemble on each machine in order to minimize setup costs while maintaining a balanced workload.** We use the estimates of production frequencies and demands, respectively, as weights in the setup cost and workload functions of the tactical planning model. We defer discussion on how to represent the setup cost and workload in order to first clarify the model's inputs, and motivate the decision problem.

We anticipate reviewing and revising the product-to-machine assignment decisions whenever the demand pattern changes or new equipment is installed (e.g., quarterly or annually). The choice of planning horizon depends on the relative stability and seasonality of demand for different products; if demand is volatile, the planning horizon is shorter. Maintaining stable medium-term product assignments has several advantages over dynamically assigning individual production batches to different machines in real-time. First, if the machines are located at different sites, then we must decide in advance which products to assemble in each machine in order to ensure proper supply chain coordination and timely deliveries. Second, as Skinner [1974] and others have suggested, creating "factories within the factory" by appropriately partitioning the products and processes reduces the complexity of managing the operations, decreases flow times, and exploits learning effects since each module specializes in a subset of products. In the electronics assembly context, adhering to fixed product-to-machine assignments offers additional advantages such as lower inventories of components and tooling, fewer component retrieval and storage transactions, lesser congestion on the floor, simpler materials handling requirements due to streamlined flow, and easier process planning. We emphasize, however, that the medium-term product assignment decisions merely provide guidelines or preferred production choices, and might only apply to a subset of regular products. In the short-term, shop floor supervisors might override the product assignment plan due to contingencies (unexpected surges in demand, rush orders, equipment downtime, and so on); similarly, the supervisors might make real-time assignment decisions for one-time (e.g., prototype boards) or low volume products based on the current status and capabilities of different modules.

We assume that the input data–product families, demand projections, production frequency estimates, and processing time–have the level of aggregation and accuracy that is

[4] Many electronics assembly facilities use periodic scheduling policies (e.g., produce high volume products once every week, low volume items once every four weeks, and so on) to facilitate due-date setting, raw materials procurement, and production scheduling. The production frequency is, therefore, a natural parameter that planners can relate to easily.

appropriate for the chosen planning horizon. For instance, computer manufacturers often produce a vast number of "part numbers", but many part numbers are minor variations or enhancements of a base model, differing only in a few model-specific components or in the programmable software. Managers typically group the individual models into a few natural product families, and prefer to use a common processing path for all members of a product family in order to simplify scheduling and exploit economies of scale. In this case, we treat each family as a single "product". This aggregation of product variations into families simplifies the demand forecasting requirements (forecasting the demand for individual product variations is often difficult and quite inaccurate), and reduces the size of the planning model. Similarly, if a subset of components always occurs as a group, then we can consider a single equivalent component type (this "component" would, of course, require multiple feeder slots on the machine).

The next two sections motivate and discuss how we represent setup cost and workload in the tactical planning model.

2.2.2 Setup management strategies

Preparing the placement machine(s) is often the major bottleneck in the changeover process (unless the total number of component types is small relative to the slot capacity). Setting up a placement machine to assemble a different product involves loading all the component types required for that product in appropriate slots on the machine, after unloading some or all of the feeders used for the previous product; we ignore other elements of the setup operation such as loading the process plan for the next product into the machine's controller. Thus, the time to setup the machine for a new batch is proportional to the number of feeders that must be loaded/unloaded. In turn, this number depends on the setup policy that the facility uses, and the bill of components for different products.

Let us first consider two extreme feeder changeover policies–*complete setup* and *incremental setup*. The **complete setup policy** involves unloading all the feeders from the machine when a batch is completed, and mounting the next product's components in appropriate slots. This policy has two advantages: (i) we can fine tune the feeder-to-slot allocation and placement sequencing decisions to minimize the cycle time per board for each product (e.g., by loading the components needed for successive placements in consecutive slots), and (ii) the process plan for a product does not change from batch to batch since each component type always has a fixed feeder location(s) for that product. However, the complete setup policy ignores the commonality of components for successive products, thus incurring long setup times.

At the other extreme, the **incremental setup policy** exploits component commonality by loading only those new component types that are not already on the machine (after selectively unloading some of the unnecessary feeders to release slots for new components). While this strategy can reduce the number of load/unload operations, its effectiveness depends on proper production sequencing (i.e., we must sequence the products to minimize sequence-dependent changover times), necessitating sophisticated scheduling methods. More importantly, since the location and composition of feeders can vary from batch to batch, the operator must dynamically decide which unnecessary feeders to unload from the machine, and which free slot to use for each new component type; in turn, the system must have the ability to dynamically change and optimize the process plan with respect to the feeder locations for the current batch. Also, the cycle time per board can degrade since the incremental setup policy permits only limited flexibility in selecting the slots for the new components, and the dynamic unloading and loading operations increase the chances of operator error (e.g., unloading the wrong feeder, loading in the wrong slot, specifying an incorrect feeder location in the process plan). Since most facilities do not currently have dynamic process planning and slot allocation capabilities, and since the degradation in cycle time might outweigh the benefits of reduced setup time, the incremental setup policy as we have described it is rarely implemented in practice.

Between these two extremes of complete and incremental setups, we might consider intermediate policies such as the group setup schemes proposed by Carmon et al. [1989] that attempt to use a common setup of components to assemble a group of similar products. When all the components (on a single side of the board) must be placed in a single pass through the placement machine, the group setup scheme corresponds to partitioning the products assigned to a machine into groups (such that the number of component types needed for each group does not exceed the slot capacity), and implementing the complete setup policy for each group. This scheme requires: (i) a methodology to group products in order to minimize setup time while ensuring that the number of component types required for all the products in each group does not exceed the machine's slot capacity (Daskin et al. [1991] propose an integer programming model and algorithm for this problem), (ii) a method to optimize the feeder locations for each group (for example, to minimize the weighted placement time for all products in the group given their relative demands), and (iii) the close coordination of production plans so that products in a group are assembled consecutively. The group setup policy is difficult to incorporate in a tactical planning model since we cannot directly express the total setup time under this policy in terms of the product-to-machine assignment decisions.

We propose the following alternate policy, which we refer to as the **partial setup policy**, that is both easy to implement and convenient to model. For a particular machine, consider the set of all component types needed to assemble the products assigned to that machine. We will partition this set into two categories: "permanent" components, and "temporary" components. The setup policy consists of loading all the permanent components in preassigned slots on the machine; we never unload these feeders except to replenish components. On the other hand, temporary components remain on the machine only when needed. Thus, to changeover from one product to another, we will unload all the temporary components corresponding to the previous product, and load the temporary components required for the next product. We have described a "complete temporary setup" version of the partial setup policy, i.e., we unload all the temporary components after completing each batch. Other variants of this policy include an incremental version that selectively unloads unnecessary temporary components, and loads only new (temporary) components during each changeover operation. Similarly, a group setup version would use a common setup of temporary components for selected groups of products (Daskin et al. [1991]). We assume that the complete temporary setup version provides an adequate approximation of the setup cost for medium-term planning purposes.

For a given assignment of products to machines, the partial setup policy requires two decisions: how many slots to dedicate for permanent components, and which components are permanent or temporary. We will incorporate both these decisions in our optimization model. Note that if the available slots on a machine can accommodate all the component types required for every product that the machine assembles, then we can designate all components as permanent; otherwise, our choice of permanent components is constrained by the requirement that, for each product, the slot capacity must equal or exceed the total number of permanent components plus the temporary components required for that product. The choice of permanent and temporary components captures the setup interactions between products that are assigned to the same machine. But, once we have classified the components as permanent and temporary, the setup time becomes separable by product (proportional to its number of temporary components), and we can develop optimized feeder locations and process plans that are invariant with production sequence. Thus, the partial setup policy provides a practical way to reap the setup time benefits due to component commonality, and modeling its cost does not require a detailed representation of product sequencing decisions (the incremental setup policy requires such a representation).

We define the **setup cost** per period of a product as the number of temporary components for that product times its production frequency. Planners might wish to use other product-specific weights (instead of production frequency) to define setup cost. If the

load/unload time is approximately the same for all components, then multiplying the setup cost by the load/unload time per feeder gives the setup time per demand period incurred to assemble the product (if different component types have different load/unload times, we can redefine the setup cost as the sum of load/unload times for all temporary components multiplied by the number of batches per period). The setup cost also serves as a surrogate metric for other important manufacturing performance indicators such as the number of materials transactions and handling operations (to retrieve and return component reels to storage), the number of spare feeders required (to prepare components for the next product), the congestion on the assembly floor (due to material movement, feeder loading operations, and reel/feeder storage), and the in-process component inventories. Minimizing setup cost, therefore, not only reduces the manhours to change over from one batch to the next but also improves several other aspects of the electronics assembly operation.

2.2.3 Balancing the workload

The total processing time per demand period to assemble a product depends on its demand, the types and number of components it requires, and the placement speed for different component types. The exact placement or cycle time per board is sensitive to the slot allocation (i.e., the relative locations of various component feeders) and placement sequencing decisions. For medium-term planning purposes, we require only a rough cut estimate of cycle time per board (including board setup time to locate fiducials, etc.) for each product on every machine, obtained using, say, average placement rates for different component types (e.g., 2 components per second for small, passive components, etc). Thus, if

d_i = demand (number of boards) per period for product i,

m_{ij} = number of type j components per board required for product i,

s_{jk} = average placement rate (number of components/unit time) for component j on machine k, and

g_{ik} = time to setup each board (or a panel of boards) on the machine, including the time to locate fiducials, and so on,

then, the cycle time per board for product i on machine k is

$$p_{ik} = \sum_j \frac{m_{ij}}{s_{jk}} + g_{ik} \text{ time units per board,}$$

and the **total processing time** to produce all d_i boards using machine k is $\mathbf{P_{ik}} = d_i p_{ik}$ time units per demand period. We might optionally add to P_{ik} an estimate (or lower bound) of the line changeover time per batch multiplied by the production frequency to account for the time to setup the screen printer and other equipment, change the reflow oven's temperature profile, and assemble and test a few pilot boards.

We define the **workload** of a machine as the sum of processing times P_{ik} for all products assigned to that machine. We model the workload balancing requirement by imposing a user-specified upper limit T_k on the total processing time for all products assigned to each machine k. The parameter T_k, which we refer to as the **processing limit**, depends on the desired machine utilization level. For instance, using performance metrics such as "placements per unit time", managers can compute the total operating hours needed to meet the projected demand for all products; allocating the total operating hours to the parallel placement machines (subject to the actual available processing time for each machine), and adding an allowance for limited imbalances, gives one set of processing limits. By parametrically varying these limits, the user can generate several (if necessary, all) pareto-optimal product assignment scenarios with different setup and workload characteristics. If the processing limits are "tight" (i.e., close to the minimum required operating time) we expect some degradation in the total setup cost and vice versa.

To summarize, our tactical planning model minimizes total setup cost for all products subject to upper limits on the processing time for each machine (workload balancing constraints) as well as component loading and slot capacity constraints. The workload balancing constraints provide a user-controllable means to influence the overall allocation of products to machines. The setup cost objective represents not only the time to changeover production from one product to another but also other factors, such as labor requirements, materials handling effort, and component inventory levels. Next, we formulate the tactical planning problem as an integer program.

2.3 Integer Programming Formulation of the Tactical Planning Problem

The tactical planning problem has two sets of decision variables: (i) the **product assignment** decisions, specifying which products to assemble on each machine, and (ii) the **component loading** decisions representing the choice of permanent and temporary components for each machine. The model has **workload balancing** constraints, **product assignment** constraints, **component loading** constraints, and **slot capacity** constraints. To formulate the problem as an integer program, we introduce the following notation.

We are given **N** products and **K** parallel machines; we use i = 1,2, ...,N and k = 1,2, ..., K, respectively, to index the products and machines. Let J = {1,2, ...,M} denote the index set of the M component types needed to assemble all products. The tactical planning model requires the following input parameters:

P_{ik} = Processing time (hours per period) for product i on machine k;
T_k = Processing limit of machine k (hours per period);
C_k = Slot capacity of machine k (number of standard feeder slots);
b_i = Anticipated production frequency of product i (number of batches per period); and,
$J(i)$ = Subset of component types required for product i.

We define $I(j)$ as the subset of products that use component j. We assume that each product i must be assembled in a single pass through the machine[5], implying that the number of component types $|J(i)|$ needed for this product must less than or equal to the slot capacity C_k for at least one machine k (otherwise, the problem is infeasible). The model's decision variables are:

$$x_{ik} = \begin{cases} 1 & \text{if we assign product i to machine k,} \\ 0 & \text{otherwise;} \end{cases}$$

$$y_{jk} = \begin{cases} 1 & \text{if we designate component j as permanent on machine k,} \\ 0 & \text{otherwise; and,} \end{cases}$$

$$z_{ijk} = \begin{cases} 1 & \text{if we load comp. } j \in J(i) \text{ temporarily on machine k to assemble prod. i,} \\ 0 & \text{otherwise.} \end{cases}$$

We can then formulate the tactical planning problem as the following integer program [P]:

$$[P] \qquad Z^* = \min \sum_{i=1}^{N} b_i \{ \sum_{j \in J(i)} \sum_{k=1}^{K} z_{ijk} \} \qquad (1.1)$$

subject to

Product assignment constraints:

$$\sum_{k=1}^{K} x_{ik} \geq 1 \qquad \text{for all } i = 1,2, ...,N, \qquad (1.2)$$

Workload balancing constraints:

$$\sum_{i=1}^{N} P_i x_{ik} \leq T_k \qquad \text{for all } k = 1,2, ...,K, \qquad (1.3)$$

---

[5] Surface mount assembly facilities prefer single-pass assembly whenever possible since permitting multiple passes increases routing complexity, and introduces additional board handling operations that can increase the defect rate (e.g., due to vibrations) and flow time. If a product requires multiple passes, we assume that the subset of components to be placed in each pass is predetermined, and so we treat each subset as a separate product.

Component loading constraints:

$$y_{jk} + z_{ijk} \geq x_{ik} \quad \text{for all } i = 1,2, ...,N,\ j \in J(i), \text{ and } k = 1,2, ...,K, \qquad (1.4)$$

Slot capacity constraints:

$$\sum_{j \in J} y_{jk} + \sum_{j \in J(i)} z_{ijk} \leq C_k \quad \text{for all } i = 1,2, ...,N, \text{ and } k = 1,2, ...,K, \text{ and} \qquad (1.5)$$

Integrality constraints:

$$x_{ik}, y_{jk}, z_{ijk} = 0 \text{ or } 1 \quad \text{for all } i = 1,2, ...,N,\ j \in J(i), \text{ and } k = 1,2, ...,K. \qquad (1.6)$$

The objective function (1.1) minimizes the total setup cost which is the weighted sum of temporary component assignments for all products. The product assignment constraints (1.2) specify that each product must be assigned to at least one machine. Since we minimize setup cost, formulation [P] must have an optimal solution that assigns each product to exactly one machine. The workload balancing constraints (1.3) impose the user-specified upper bounds T_k on the total processing time assigned to each machine k. For every product i, the component loading constraints (1.4) ensure that, if we assign product i to machine k (i.e., if $x_{ik} = 1$), then each of its components $j \in J(i)$ must be loaded on machine k either permanently or as a temporary component for that product, i.e., we must set either $y_{jk} = 1$ or $z_{ijk} = 1$. The slot capacity constraint (1.5) ensures that the component loading decisions are feasible; it specifies that, for every product i and machine k, the number of slots reserved for permanent components (the first term in the left-hand side of (1.5)) plus the number of temporary components required for product i (the second term in the left-hand side, which is zero if the optimal solution does not assign product i to machine k) must not exceed the machine's slot capacity C_k.

2.3.1 Model variants

By making minor changes to formulation [P] we can model several variants of the tactical planning problem. For instance, to permit **product splitting** (i.e., assigning a product to more than one machine) we can either relax the integrality constraints on the product assignment variables x_{ik}[6] (in which case x_{ik} denotes the fraction of product i's demand that is assigned to machine k), or define multiple products whose total demand equals the original product's demand. To incorporate **predetermined assignments** of certain products to machines, e.g., if product i is preassigned to machine k due to its special processing

---

[6] We retain the integrality restrictions on the y and z variables.

requirements, we can either add the constraint $x_{ik} = 1$ to formulation [P] or eliminate the variables x_{ik}, for all k = 1,2, ...,K, and subract product i's processing time P_{ik} from machine k's processing limit T_k. Similarly, adding the **joint assignment** constraints $x_{ik} = x_{i'k}$ or the **exclusive assignment** constraints $(x_{ik} + x_{i'k}) \leq 1$ for all machines k = 1,2, ...,K, models the conditions that products i and i' must or must not be assigned to the same machine. We can incorporate **component-dependent** setup costs and feeder widths by changing the coefficients in the objective function and the slot capacity constraints. The model also applies to placement modules that contain **more than one machine in series.** In this case, the slot capacity of the module is the combined slot capacity of all the machines in series, and the processing time P_{ik} equals product i's demand times the cycle time per board for product i in module k[7]. Finally, our model treats the number of slots allocated to permanent components as a (implicit) decision variable. Alternatively, we can exogenously set the number of permanent slots or specify an upper limit, say, U_k for the number of permanent components on machine k by adding the constraints:

$$\sum_{j \in J} y_{jk} \quad \leq \quad U_k \qquad \text{for all } k = 1,2, ...,K.$$

By solving the model for different values of U_k, we can assess the sensitivity of setup cost to the permanent slot reservation decisions. Our solution methodology extends to all these model variants; however, we will consider only the basic model [P] in our subsequent discussions.

2.3.2 Special cases

Formulation [P] also has several interesting special cases. If the slot capacities C_k are very large (e.g., $C_k \geq M$ for all machines k) so that we can load all the required components permanently on each machine, and if all the machines have the same processing limit, say, T, then problem [P] reduces to the following recognition version of the nonpreemptive, parallel machine scheduling problem: is there an assignment of products to machines such that the makespan is less than or equal to T? Even with identical machines (i.e., each product has the same processing time on every machine), this problem is NP-complete (see, for instance, Lawler, Lenstra, and Rinnooy Kan [1982]).

Another interesting special case arises when we choose large values (say, greater than the sum of the processing times for all the products) for the processing limits T_k. In this case, we seek the best grouping of the N products into at most K groups such that the total setup

---

[7] If the module contains L machines in series, the smallest possible cycle time per board (achieved when all the machines are equally utilized) is 1/L times the total placement time per board. To estimate the actual cycle time per board, we might inflate this lower bound to account for possible variations in the placement time of each machine.

cost is minimized. Unlike traditional group technology approaches based, for instance, on similarity indices (see, for example, DeWitte [1980], Rajagopalan and Batra [1982]) or rank order clustering (e.g., King [1980] and King and Nakornchai [1982]) that use component commonality as a surrogate for setup cost, the optimization model [P] provides a more complete representation (including an explicit setup cost function) of the grouping problem by accounting for the limited slot capacity of each machine, and the production frequency of each product. Of course, we might consider adapting the group technology approach, combined with parallel machine scheduling methods, to solve problem [P] heuristically.

2.3.3 Subproblems

We now consider two single-machine subproblems of [P]–the product selection problem and the setup optimization problem–that are central to our solution strategy. Suppose each product i generates a "profit" (or opportunity cost) if we decide to assemble it. The *net profit* for any subset of products is the total profit for all products in this subset minus the total setup cost to assemble the products on a single machine. The following **product selection problem** determines which products to accept for assembly on the machine:

> Given N available products, each with a profit value, select a subset of products to assemble on a single machine in order to maximize net profit, subject to the workload balancing constraint (1.3), component loading constraints (1.4), and slot capacity constraints (1.5).

This product selection problem is relevant for surface mount assembly subcontractors or "profit center" line managers who must decide which of several available products to accept for assembly.

For a given assignment of products to a machine, we refer to problem of selecting permanent and temporary components as the **setup optimization** problem, and define it as follows:

> Given the set of products to be assembled on a machine, decide which components must be loaded permanently in order to minimize the total setup cost for the remaining temporary components subject to the component loading constraints (1.4) and the slot capacity constraints (1.5).

This model has direct application for short-term process planning, and is also a subproblem of the product selection problem. As part of our algorithmic development in Section 3, we describe solution procedures for both these subproblems.

3. Solving the Tactical Planning Problem

Formulation [P] is a large-scale integer program that is difficult to solve optimally using general-purpose integer programming methods. As we show in Section 3.2, even the setup optimization subproblem is NP-hard. We, therefore, focus on developing an optimization-based procedure that exploits the problem's embedded special structure to construct a good feasible solution, and also generates a lower bound to assess the quality of this solution. Our solution method combines column generation with greedy and local improvement heuristics, and other lower bounding techniques.

Before describing the column generation algorithm, we note that the linear programming relaxation of formulation [P] can have arbitrarily poor performance, i.e., the gap between the optimal setup cost and LP value, as a proportion of the LP value, can be arbitrarily large. For instance, suppose all K machines are identical, i.e., they all have the same slot capacity C, processing limit T, and processing times P_i, and suppose the number of component types M is less than or equal to K C. Then, the following fractional solution solves the LP relaxation of formulation [P] optimally:

$$x_{ik} = \frac{1}{k} \qquad \text{for all } i=1,2, \ldots,N, \text{ and } k=1,2, \ldots,K,$$

$$y_{jk} = \frac{1}{k} \qquad \text{for all } j \in J, \text{ and } k=1,2,\ldots,K, \text{ and}$$

$$z_{ijk} = 0 \qquad \text{for all } i=1,2, \ldots,N, j \in J(i), \text{ and } k=1,2, \ldots,K.$$

This solution assigns M/K permanent components to each machine, which by assumption does not exceed the slot capacity C. For the original problem to be feasible, the processing limit T must exceed 1/Kth the total processing time required to assemble all the products; hence, the LP solution satisfies the workload balancing constraint. Since the LP solution does not require any temporary components, its objective function value is zero; however, we can make the optimal setup cost arbitrarily large by considering products with low component commonality and high values for the production frequency parameters b_i.

Since formulation [P] has a very weak LP relaxation, solving it using a standard LP-based branch-and-bound method is not likely to be very effective. Instead, we reformulate the problem as a capacitated set covering model, and solve its LP relaxation using a column generation algorithm (see, for instance, Lasdon [1972] or Bradley, Hax, and Magnanti [1977]). Column generation have proved to be an effective technique for solving difficult crew scheduling, cutting stock, and vehicle routing problems (e.g., Barnhart et al. [1992], Vance et al. [1992], and Desrochers et al. [1992]). For the tactical planning model, the

column generation procedure iteratively solves the product selection subproblem heuristically and/or using branch-and-bound. The method generates lower bounds on the optimal setup cost, and also enables us to construct good feasible solutions. The next three sections describe the various components of our algorithm to approximately solve the tactical planning problem. Section 3.1 reformulates the tactical planning problem as a capacitated set covering problem, and provides an overview of the solution approach. Sections 3.2 and 3.3 discuss solution procedures for the two subproblems–the product selection and setup optimization subproblems–that we must solve in order to dynamically generate columns for the set covering formulation. Section 3.4 integrates these subproblem solution methods in the overall algorithmic framework, and discusses our heuristic and lower bounding strategies.

3.1 Overview of the Solution Approach

Let us first reformulate the tactical planning problem. Suppose we can enumerate all the feasible assignment patterns for each machine. A *feasible assignment pattern* (which we often abbreviate as "pattern") for machine k is a subset of products whose total processing time on machine k does not exceed the machine's processing limit T_k. Let H_k be the total number of feasible patterns for machine k. For h = 1,2, ...,H_k, let $\bar{a}_{hk}$ be the characteristic vector or "column" corresponding to the h^{th} pattern for machine k, with elements $a_{ihk} = 1$ if product i is assigned to machine k in this pattern, and 0 otherwise. Let s^*_{hk} denote the minimum setup cost for the h^{th} pattern, obtained by optimally solving the setup optimization problem assuming that all products i with $a_{ihk} = 1$ are assigned to machine k.

We can interpret the tactical planning problem as the process of choosing K patterns, one for each machine, such that every product is included in at least one of the chosen patterns. Using binary decision variables λ_{hk} to represent the choice of patterns (λ_{hk} is 1 if we choose the h^{th} pattern for machine k, and is 0 otherwise), we can formulate the tactical planning problem as the following set covering model with side constraints, which we denote as [P']:

$$\textbf{[P']} \qquad Z^* = \min \sum_{k=1}^{K} \sum_{h=1}^{H_k} s^*_{hk} \lambda_{hk} \qquad (3.1)$$

subject to

Product covering constraints:

$$\sum_{k=1}^{K} \sum_{h=1}^{H_k} a_{ihk} \lambda_{hk} \geq 1 \qquad \text{for all } i = 1,2, ...,N, \qquad (3.2)$$

Pattern selection constraints:

$$\sum_{h=1}^{H_k} \lambda_{hk} \leq 1 \quad \text{for all } k = 1,2, ...,K, \text{ and} \qquad (3.3)$$

Integrality constraints:

$$\lambda_{hk} = 0 \text{ or } 1 \quad \text{for all } k = 1,2, ...,K, \text{ and all } h = 1,2, ...,H_k. \qquad (3.4)$$

The objective function (3.1) minimizes the total setup cost of all selected patterns. The product covering constraints (3.2) ensure that every product belongs to at least one chosen pattern. Constraint (3.3) specifies that we cannot choose more than one pattern for each machine.

To simplify our subsequent discussions, we will **assume that all K machines are identical**, i.e., they each have the same slot capacity C, processing times P_i, and processing limit T. The methodology extends easily to problems with non-identical machines (including modules containing multiple placement machines in series) although we have not tested it for these problems. When the machines are identical, an assignment pattern that is feasible for one machine must also be feasible and have the same setup cost for all other machines. Hence, we can drop the subscript k for the problem parameters and decision variables in formulation [P'], and combine the K pattern selection constraints (3.3) into a single "capacity" constraint. Let H denote the total number of feasible assignment patterns. Performing these simplifications, we get the following **Capacitated Set Covering** formulation, denoted as **[CSC]**, for the electronics assembly tactical planning problem with identical parallel machines.

[CSC]

$$Z^* = \min \sum_{h=1}^{H} s_h^* \lambda_h \qquad (3.5)$$

subject to

Product covering constraints:

$$\sum_{h=1}^{H} a_{ih} \lambda_h \geq 1 \quad \text{for all } i = 1,2, ...,N, \qquad (3.6)$$

Pattern selection constraint:

$$\sum_{h=1}^{H} \lambda_h \leq K, \text{ and} \qquad (3.7)$$

Integrality constraints:

$$\lambda_h = 0 \text{ or } 1 \quad \text{for all } h = 1,2, ...,H. \qquad (3.8)$$

As before, the binary variable λ_h denotes whether ($\lambda_h = 1$) or not ($\lambda_h = 0$) we choose pattern h, and constraints (3.6) ensure that every product belongs to at least one chosen pattern. We no longer associate a pattern with a particular machine, i.e., we can arbitrarily assign the selected patterns to the available machines; constraint (3.7) limits the number of chosen patterns to K since we cannot assign more than one pattern to each machine.

Let [LCSC] denote the linear programming relaxation of the integer program [CSC], obtained by replacing the integrality restrictions (3.8) with nonnegativity constraints. Since formulation [CSC] contains a very large number of columns (exponential in the number of products N), enumerating all the columns and solving the integer program optimally is impractical. Therefore, we: (i) solve its LP relaxation [LCSC] using an iterative column generation procedure to generate lower bounds, and (ii) apply heuristic and optimization procedures to generate good feasible solutions. We refer to the linear program [LCSC], containing all the columns of [CSC], as the *master problem*, a restricted version of [LCSC] containing only a subset of its columns is called a *restricted master problem* . Our solution procedure consists of the following three phases:

Phase 1: Initial Heuristic Solution

Using some greedy and local improvement heuristics we find K (or fewer) feasible assignment patterns that together cover all the products. We refer to this solution as the *initial heuristic solution.*

Phase 2: Solving the LP relaxation of [CSC]

The optimal value [LCSC] is a lower bound on the optimal setup cost Z^*. We attempt to solve this LP relaxation optimally via column generation. Starting with the columns defined by the initial heuristic solution, the method iteratively generates additional columns by solving the product selection subproblem; this subproblem uses "profit" values derived from the optimal dual solution to the restricted master problem containing only the columns generated in previous iterations.

This phase provides:

- feasible solutions at intermediate iterations whenever the optimal LP solution to the restricted master problem is integral. We refer to these solutions as *intermediate feasible solutions*;
- *intermediate lower bounds* on the minimum total setup cost, based on the optimal dual solution to the restricted master problem at intermediate iterations, and
- a *final lower bound* if we solve [LCSC] optimally.

Phase 3: Final Feasible Solution

Since the column generation procedure in Phase 2 generates many promising assignment patterns, we construct a *final feasible solution* by optimally solving the restricted CSC integer program using all the available columns.

The first two phases require solving the setup optimization subproblem numerous times (for every candidate assignment pattern). We first discuss this subproblem and its solution method before returning to the overall solution procedure.

3.2 The Single Machine Setup Optimization Subproblem

The **Setup Optimization** (SO) subproblem is a core model for both the product selection problem and the original tactical planning model. For a given assignment of products to a machine, the SO model determines the number of permanent slots and selects the components to be loaded in these slots subject to the slot capacity constraints, in order to minimize the setup cost for the remaining "temporary" components. This section first simplifies the formulation of the SO problem, and shows that this problem is NP-hard. We then develop an upper bound on the number of permanent components in any feasible solution. Adding an explicit constraint to enforce this upper bound strengthens the linear programming relaxation of the SO model. We also describe a heuristic solution procedure, develop a lower bounding method, and propose a branching scheme to solve the problem optimally.

3.2.1 Notation and problem formulation

Consider any feasible assignment pattern $\bar{a}$ that assigns the product subset $I(\bar{a})$ to a single machine. To simplify the notation, we omit the argument $\bar{a}$, and let I now denote the subset of products selected by pattern $\bar{a}$, i.e., $\mathbf{I} = \{i = 1,2,...,N: a_i = 1\}$. Similarly, **J** is the set of all components required to assemble products in this subset, and $I(j) \subseteq I$ denotes the subset of products requiring component j. Since the given assignment pattern is feasible, the total processing time for all the products in I is less than or equal to T. The SO problem contains the component loading, slot capacity, and integrality constraints of formulation [P], but without the machine index k since we now consider only a single machine; we restate the SO formulation for convenience. The model contains two sets of binary decision variables z_{ij} and y_j, for every product $i \in I$ and all components $j \in J$. The decision variable y_j takes the value 1 if we designate component j as a permanent component, and is 0 otherwise; z_{ij} is 1 if component j must be loaded temporarily for product i, and 0 otherwise. Using these decision variables, the SO problem has the following formulation:

[SO'] $$s^*(\bar{a}) = \min \sum_{i \in I} b_i \{ \sum_{j \in J(i)} z_{ij} \} \quad (3.9)$$

subject to

Component loading constraints:

$$y_j + z_{ij} \geq 1 \quad \text{for all } i \in I, \text{ and all } j \in J(i), \quad (3.10)$$

Slot capacity constraints:

$$\sum_{j \in J} y_j + \sum_{j \in J(i)} z_{ij} \leq B \quad \text{for all } i \in I, \text{ and} \quad (3.11)$$

Integrality constraints:

$$y_j, z_{ij} = 0 \text{ or } 1 \quad \text{for all } i \in I, \text{ and all } j \in J(i). \quad (3.12)$$

In the capacitated set covering formulation [CSC], the objective coefficient for the pattern selection variable λ_h corresponds to the minimum setup cost $s^*(\bar{a}_h)$ obtained by solving [SO'] for the h^{th} pattern $\bar{a}_h$.

Since the production frequency parameters b_i are non-negative, formulation [SO'] has an optimal solution that designates each component $j \in J$ as either permanent or temporary but not both, i.e.,

$$z_{ij} = 1 - y_j \quad \text{for all } i \in I, \text{ and all } j \in J(i). \quad (3.13)$$

Using (3.13) we can eliminate the z variables from formulation [SO'] to get the following more compact formulation [SO]:

[SO] $$s^*(\bar{a}) = \sum_{j \in J} \sum_{i \in I(j)} b_i - \max \sum_{j \in J} (\sum_{i \in I(j)} b_i) y_j \quad (3.14)$$

subject to

$$\sum_{j \notin J(i)} y_j \leq C - |J(i)| \quad \text{for all } i \in I, \text{ and} \quad (3.15)$$

$$y_j = 0 \text{ or } 1 \quad \text{for all } j \in J. \quad (3.16)$$

The objective function now maximizes the savings in setup cost obtained by assigning components permanently. The slot capacity constraints (3.15) state that, for each product i, the number of slots occupied by permanent components that are <u>not</u> needed for product i must be no greater than the remaining slot capacity after loading all of product i's components.

3.2.2 Computational complexity of the SO problem

Proposition 1: The setup optimization problem is NP-hard.

Proof:
We will prove this result by transforming any instance of the following recognition version [SP] of the set packing problem into a recognition version of formulation [SO], which we refer to as the equivalent SO problem, in polynomial time.

Set Packing problem: [SP]

Given a set $E = \{1,2, ..., |E|\}$ of elements, and integer $L \leq |E|$, and P subsets $S_p \subseteq E$, for $p = 1,2,...,P$, is there a Boolean vector $\bar{x} \in \{0,1\}^P$ satisfying the following conditions?

$$\sum_{p=1}^{P} x_p \geq L, \text{ and} \qquad (3.17)$$

$$\sum_{p:e\in S_p} x_p \leq 1 \qquad \text{for all } e \in E. \qquad (3.18)$$

Given any instance of [SP], we will consider an equivalent SO problem with $N = |E|+1$ products and $M \geq 2P$ components. The first |E| products are **regular products**, indexed from 1 to |E|, one corresponding to each element of the set E, and product N is a **dummy product**. We define the component set J as $J' \cup J''$, where $\mathbf{J'} = \{1,2, ..., P\}$ is the set of **regular components**, containing one component p corresponding to each subset S_p in the original SP problem instance, and **J"** is a set of **special components** that we add in order to ensure that every product requires the same number of component types. We construct the set J" as follows.

For every regular product $i = 1,2, ..., |E|$, let α_i denote the number of subsets S_p that contain the i^{th} element of E in the given SP instance. For each regular product i, we create $(P-\alpha_i)$ special components; let J''_i denote this set of special components. If $\{P + \sum_{i=1}^{|E|} (P-\alpha_i)\} < 2P$, we create $P - \sum_{i=1}^{|E|}(P-\alpha_i)$ additional special components. The set J" contains all the special components created in this manner; these components are indexed consecutively from (P+1). Note that by construction, the total number of components M (= |J'| + |J"|) in the equivalent SO problem instance is greater than or equal to 2P.

For every **regular product** i = 1,2, ...,|E|, we define the set of components J(i) that it requires as follows:

$$J(i) = \{j \in J' : i \notin S_j\} \cup \{J'' \setminus J''_i\}.$$

The **dummy product** requires every regular component, but does not use any special component, i.e., J(N) = J'. Observe that |J(i)| = M – P for i = 1,2, ...,|E|, and |J(N)| = P. We set the slot capacity C of the machine equal to M – P + 1. Since C ≥ |J(i)| for all products i = 1,2, ...,N, the SO problem is feasible.

The production frequency b_i is 1 for every regular product i = 1,2, ...,|E|, but the dummy product has b_N = MN+1. The equivalent SO problem, which is a recognition version of formulation [SO], seeks a choice of permanent components (i.e., y_j values for all components j ∈ J) such that the total setup savings is at least L b_N = L(MN+1), i.e., a solution $\bar{y}$ is feasible for the equivalent SO problem if it satisfies constraints (3.15), (3.16), and the following **minimum setup savings constraint**:

$$\sum_{j \in J} \sum_{i \in I(j)} b_i y_j \geq L(MN+1). \qquad (3.19)$$

We claim that [SP] has a feasible solution if and only if the equivalent SO problem has a feasible solution. We will establish this claim by showing that given a feasible solution to either problem we can construct a feasible solution to the other problem. For any feasible SP solution $\bar{x}$, we construct the *corresponding SO solution* by setting $y_p = x_p$ for all the regular components p = 1,2, ...,P, and $y_j = 0$ for all the special components j > P. Conversely, given a feasible solution $\bar{y}$ to the equivalent SO problem, we set $x_p = y_p$ for all p = 1,2, ...P to get the *corresponding SP solution.*

First, note that, in the equivalent SO problem the savings for selecting a regular component j is between (MN+1) and (MN+1+N–1), while the savings for each special component is at most (N–1) since: (i) each of the (N–1) regular products i has b_i = 1, (ii) the dummy product has b_N = MN+1, and (iii) the dummy product requires every regular component but no special component. Ignoring the dummy product, the maximum possible savings even if we can select all the regular and special components as permanent is (N–1)(M–P) which is less than b_N =MN+1. Therefore, in order to satisfy the minimum setup savings constraint (3.19), every feasible solution $\bar{y}$ to the equivalent SO problem must capture the dummy product's savings for at least L permanent components, i.e., it must select at least L regular components. Hence, the corresponding SP solution satisfies (3.17). Conversely, since every feasible SP solution $\bar{x}$ sets at least L of the x_p variables to 1, the savings for the corresponding SO solution exceeds L(MN+1), i.e., this solution satisfies the minimum setup savings constraint (3.19).

We now show that the corresponding SO and SP solutions satisfy constraints (3.15) and (3.18) respectively. Note that the slot capacity constraint (3.15) for every regular product i in our equivalent SO problem instance becomes

$$\begin{aligned} \sum_{p \in J' : i \in S_p} y_p + \sum_{j \in J'' : j \notin J(i)} y_j &\leq C - |J(i)| \\ &= (M-P+1) - (M-P) \\ &= 1 \end{aligned} \qquad (3.20)$$

for all i = 1,2, ...,|E|. The slot capacity constraint for the dummy product specifies that the SO solution must select no more than B–|J(N)| = (M–P+1) – P special components. Therefore, since every feasible SP solution $\bar{x}$ satisfies constraints (3.18), the corresponding SO solution must satisfythe slot capacity constraints (3.15). Similarly, for any feasible solution $\bar{y}$ to the equivalent SO problem, the corresponding SP solution satisfies constraints (3.18).

These arguments show that [SP] has a feasible solution if and only if the equivalent SO problem has a feasible solution. Since the set packing problem is NP-complete, the setup optimization problem [SO] is NP-hard. ♦

3.2.3 Greedy heuristic for the SO problem

Although the SO problem is intractable in theory, our computational experiments indicate that its linear relaxation provides a tight lower bound, and the following adaptation of the greedy heuristic described in Nemhauser and Wolsey [1988] for the set packing problem generates good component loading plans. We define $SC_j = \sum_{i \in I(j)} b_i$ as the *setup cost for component j*. SC_j is the total setup cost we incur if component j is temporary (recall that, in an optimal solution, any component that is loaded temporarily for one product must be temporary for all the products that use it). The heuristic seeks to maximize the setup cost savings at each step by permanently loading components j that have high values of $\sigma_j = SC_j/(N-|I(j)|)$. The denominator of σ_j represents the number of rows (constraints (3.15)) covered by component j in formulation [SO]. Hence, the parameter σ_j represents the *setup savings per row covered* if we assign a permanent slot to component j. Starting with no permanent components, the procedure successively examines every component j in decreasing order of σ_j (in case of ties, we give priority to components with higher values of SC_j), and makes component j permanent if feasible, i.e., if setting $y_j = 1$ and $y_{j'} = 1$ for all previously chosen permanent components j' does not violate the slot capacity constraint (3.15) for any product i. Using a worst-case example similar to the set packing example in

Nemhauser and Wolsey [1988], we can show that the performance of this greedy SO heuristic can be arbitrarily bad relative to the optimal value. However, the method seems to perform quite well in practice. Next, we describe a method to generate a lower bound on the optimal setup cost s_h^*.

3.2.4 Lower bound on optimal setup cost

Let Q_{min} denote the minimum number of slots required for temporary components in order to ensure feasibility of the slot capacity constraints for all products. $R_{max} = C - Q_{min}$ is the maximum number of permanent slots in any feasible component loading scheme. We develop a lower bound on the minimum setup cost by first determining a **lower limit Q** on Q_{min}. Then, $R = C - Q$ is an upper bound on R_{max}, and in the best possible scenario we can choose the R components with the largest setup costs SC_j as permanent components. Hence, the minimum setup cost on the machine must equal or exceed the sum of setup costs for the remaining (M–R) components.

For any given subset of products $S \subseteq I$, let $\mathbf{J(S)} = \bigcup_{i \in S} J(i)$ be the set of all components required to assemble the products in S. To compute Q, we use the following proposition.

Proposition 2:

$$Q_{min} \geq \left\lceil \frac{|J(S)|-C}{|S|-1} \right\rceil \text{ for all subsets of boards } S \subseteq I \text{ with } |S| \geq 2. \qquad (3.21)$$

Proof:

Let $l = |S|$, and without loss of generality assume $S = \{1,2,...,l\}$. First, we prove by induction on l that every feasible SO solution must satisfy

$$l\, Q_{min} \geq |J(S)| - \sum_{j \in J(S)} y_j. \qquad (3.22)$$

Consider a subset S containing only two products, say, $S = \{1,2\}$. The number of temporary slots must be greater than or equal to the number of temporary components for each product. Hence,

$$Q_{min} \geq |J(i)| - \sum_{j \in J(i)} y_j \quad \text{for all } i \in I. \qquad (3.23)$$

Summing inequalities (3.23) for i = 1,2 gives

$$\begin{aligned} 2Q_{min} &\geq |J(1)| + |J(2)| - \sum_{j \in J(1)} y_j - \sum_{j \in J(2)} y_j \\ &= |J(1) \cup J(2)| + |J(1) \cap J(2)| - \sum_{j \in J(1) \cup J(2)} y_j - \sum_{j \in J(1) \cap J(2)} y_j \end{aligned}$$

$$= |J(S)| + |J(1) \cap J(2)| - \sum_{j \in J(S)} y_j - \sum_{j \in J(1) \cap J(2)} y_j$$

$$\geq |J(S)| - \sum_{j \in J(S)} y_j \qquad \text{since } y_j \leq 1 \text{ for all } j \in J.$$

Hence, every subset S containing two products satisfies inequality (3.22).

Now, suppose that (3.22) is valid for all subsets containing (l–1) or fewer products ($l \geq 3$). Consider a subset S = {1,2,...,l} containing l products, and let S' = {1,2,...,l-1}. Adding the inequality (3.22) corresponding to subset S', and the inequality (3.23) for product l gives

$$l\, Q_{min} \geq |J(S')| + |J(l)| - \sum_{j \in J(S')} y_j - \sum_{j \in J(l)} y_j$$

$$= |J_S| + |J(S') \cap J(l)| - \sum_{j \in J(S)} y_j - \sum_{j \in J(S') \cap J(l)} y_j$$

$$\geq |J(S)| - \sum_{j \in J(S)} y_j \qquad \text{since } y_j \leq 1 \text{ for all } j \in J.$$

Therefore, (3.22) is a valid inequality for all subsets $S \subseteq I$ with $|S| \geq 2$. Note also that

$$C \geq Q_{min} + \sum_{j \in J} y_j$$

$$\geq Q_{min} + \sum_{j \in J(S)} y_j.$$

Adding this inequality to (3.22) we get

$$(l - 1)\, Q_{min} \geq |J(S)| - C. \qquad (3.24)$$

Dividing both sides of (3.24) by (l – 1), and rounding up the right-hand side (since Q_{min} is an integer) shows that (3.21) is a valid inequality for the SO problem. ♦

Proposition 2 suggests one way to determine the lower bound Q on the minimum number of temporary components Q_{min}: compute $Q(S) = \lceil (|J(S)|-C)/(|S|-1) \rceil$ for every subset S (where $\lceil g \rceil$ denotes the smallest integer greater than or equal to g), and set Q equal to the maximum value over all subsets. Note that this approach is valid even if we do not completely enumerate all the subsets S, i.e., the maximum value of Q(S) over any limited family of subsets S also provides a valid lower bound on Q_{min}. We exploit this observation in two of the following three lower bounding procedures.

Methods to compute the lower bound Q

Each of the following three methods generates a valid lower bound on Q_{min}. In our computational tests, none of these three methods systematically outperforms the others in

terms of producing a superior lower bound for Q_{min}; so we apply all three methods and set Q equal to the maximum lower bound.

1. The *all pairs method* computes Q(S) for all pairs of products (i.e., for all S with |S| = 2), and selects the maximum value as the lower bound.

2. The *maximal set method* attempts to identify a subset S requiring the maximum number of components |J(S)|. Initially, the set S_N contains all the products. Then, for l = N–1, ...,2, we obtain a subset S_l containing l elements by dropping from the current subset S_{l+1} the product that results in the smallest reduction in $|J(S_l)|$ (we break ties by dropping the product with the fewest components). The maximum value of Q(S) for these N–1 subsets provides a lower bound on Q_{min}.

3. For any trial value of Q, the *infeasibility method* attempts to show that the problem instance cannot have a feasible solution with only Q (or fewer) slots devoted to temporary slots. Starting with a trial value of Q = 0, the method successively increases Q until it can no longer prove the infeasibility of the current trial value. We say that the current value Q is **infeasible** if we can show that having just Q temporary slots is inadequate. On the other hand, the current value Q is **acceptable** if we cannot prove that it is infeasible. To prove that the trial value of Q is infeasible, the procedure first applies a *permanent slot reservation test* to underestimate the fewest number of permanent slots needed to assemble all products in I if only Q slots are temporary. The test fails to prove that Q is infeasible if the underestimate is less than or equal to C–Q (which is the number of permanent slots available if Q slots are temporary). If the underestimate exactly equals C–Q, we try to prove the infeasibility of Q by applying a second test, the*permanent component identification test*, which identifies specific components that must necessarily be permanent if Q slots are temporary. Appendix A contains the detailed description of the infeasibility method.

   To apply the permanent slot reservation test (Step 2), the procedure selects the available product $i^* \in I'$ that has the maximum number of remaining components, say, m_{i^*} (belonging to the current component set J'). If m_{i^*} exceeds the trial value of Q then at least (m_{i^*} – Q) components of product i* must be loaded permanently. Hence, we reduce the available permanent slot capacity γ by this amount, remove product i* from I', eliminate all its components from the current component set J', and repeat Step 2. If the number of slots that we have reserved for the products in I' exceeds the available permanent capacity (i.e., if $\gamma < 0$), then we cannot satisfy the slot capacity constraints (3.15) for all the products if we limit the number of temporary components to a

maximum of Q. In this case, we increment the value of Q by 1 (in Step 4), and reapply the infeasibility checking procedure. If m_{i*} is less than or equal to Q in Step 2b, then the current value of Q appears to be large enough and the procedure terminates (Step 5). Finally, if the number of available permanent slots (C–Q) is just enough (i.e., $\gamma = 0$ in Step 2c and the subsets of remaining products and components are not empty), then we apply the permanent component identification test (Step 3). At the end of Step 2, consider any remaining product $i \in I'$ that requires exactly Q components from J'; every other component $j \in J(i) \backslash J'$ needed for product i must be made permanent. Using this principle for all products $i \in I'$, we build the set J" of permanent components (in Step 3c). If |J"| exceeds the available permanent capacity (C–Q), then the current value of Q temporary slots is infeasible, and as before we increment Q (in Step 4) and reapply the two infeasibility tests. If the procedure reaches Step 5, we cannot prove that the current value of Q is infeasible, and so we use this value as the lower bound on the minimum number of temporary slots required.

Observe that if we apply the all pairs heursitic first, we can stop the search in the maximal set heuristic at size $l = 3$, and then use the better of the bounds produced by these two methods as the initial trial value in the infeasibility method. Our implementation follows this strategy.

The sum of setup costs SC_j for all components except the first $R = C - Q$ components with the highest setup costs gives a valid lower bound on the minimum setup cost s_h^* of the SO problem. We can also add the following *maximum permanent slots* constraint

$$\sum_{i=1}^{N} y_j \leq R \qquad (3.25)$$

to formulation [SO] in order to strengthen its relaxation. Adding this valid inequality raises two interesting issues:

(i) Does constraint (3.25) eliminate any fractional solutions to the LP relaxation of formulation [SO]?

(ii) Does this constraint define a high dimension face of the slot allocation polyhedron $conv(\{y \in \{0,1\}^{|J|} : \sum_{j \notin J(i)} y_j \leq C - |J(i)| \text{ for all } i \in I\})$?

We can construct problem instances for which adding constraint (3.25) eliminates the fractional LP solution to [SO]. The example described in Vanderbeck [1993], but with a slot capacity of 6, has this property. For this example, the upper bound R equals the true maximum value R_{max} of the number of permanent slots and the optimal LP solution to the strengthened formulation is integral. Vanderbeck [1993] also shows that: (a) using up the

maximum number of permanent slots is not always optimal, and (b) in general, the face defined by (3.25) is not a facet of the setup optimization polyhedron.

This section has focussed on the setup optimization model, which is a core subproblem to generate lower bounds as well as heuristic solutions for the tactical planning problem. We simplified the SO problem formulation, showed that it is NP-hard, described a greedy heuristic to generate a feasible slot allocation, and developed a lower bound for the setup cost based on an estimate of the maximum number of permanent slots. If this lower bound equals the cost of the heuristic slot allocation, then the heuristic solution is optimal. Otherwise, to solve the problem optimally, we use a branch-and-bound scheme. The greedy heuristic solution serves as the initial incumbent in this procedure, and we incorporate the maximum permanent slots constraint (3.25) in the SO problem formulation in order to improve the intermediate LP-based lower bounds. In our computational experience, we found that adding a new variable W, denoting the number of permanent slots, with the defining constraint $W = \sum_{j \in J} y_j$, and branching first on this variable improves the performance of the branch-and-bound scheme. The size of the branch-and-bound tree tends to be quite small (no branching was necessary for over 95% of the SO problems we solved) since the LP relaxation of the enhanced formulation (with inequality (3.25)) produces tight bounds.

In the next few sections, we describe upper and lower bounding techniques for the higher level problems–the product selection subproblem, and the overall tactical planning model.

3.3 Initial Product Assignment Heuristic

Phase 1 of our overall solution procedure for the tactical planning problem requires finding a heuristic assignment of products to machines. We apply two methods and select the better of the two solutions as our initial heuristic solution. Both these methods require repeatedly solving (approximately or optimally) the setup optimization subproblem.

3.3.1 List processing heuristic

This method successively assigns products, in decreasing order of their maximum potential setup cost $b_i |J(i)|$, to machines so as to minimize the incremental setup cost at each step. We use a slightly different ranking of products in the first K steps in order to heuristically differentiate the product groups that are assigned to each of the K machines.

Assigning the first product to each machine:

The first product assigned to a machine serves as a "seed" for similar products that are subsequently assigned to that machine based on incremental setup cost considerations (which depends on part commonality). To heuristically reduce the overlap in the set of components needed for the K final product groups, we assign "dissimilar" seed products to the K machines using the following procedure. We begin by assigning to the first machine the product with the maximum potential setup cost. At every subsequent step k, for k = 2, ...,K, we recompute the maximum potential setup cost for each of the remaining products, ignoring the components required by the first (k–1) products, and assign the product with the largest (potential) cost to the k^{th} machine.

Assigning the remaining products:

We then sort the remaining products in increasing order of $b_i |J(i)|$, and assign them in sequence to the machine that requires the smallest incremental setup cost. Ties are broken according to the longest processing time rule since this rule has proven effective for minimizing makespan, which is an implicit objective to meet the workload balancing constraints. Consider an intermediate step when product i is the next product to be assigned from the sorted list. Let I_k denote the current set of products assigned to machine k; J_k is the set of components needed to assemble these products. We define the incremental cost of assigning product i to machine k is the increase in setup cost on machine k when we add product i to I_k. If $J(i) \subseteq J_k$, or if $|J_k \cup J(i)| \leq C$, then the incremental setup cost for product i on machine k is 0. Otherwise, we must solve the SO problem assuming that machine k assembles every product in the set $I_k \cup \{i\}$. Let TS_1 and TS_2 denote, respectively, the optimal total setup cost on machine k assuming that this machine assembles the product subsets I_k and $I_k \cup \{i\}$. If we assign product i to machine k, then $TS_2 - TS_1$ is a lower bound on the actual setup cost for product i in the final solution. On the other hand, if we solve the SO problem only heuristically, then we get upper bounds on TS_1 and TS_2. We use the heursitic incremental cost, which is the difference in the heuristic setup costs for I_k and $I_k \cup \{i\}$, as an approximation for product i's actual setup cost if it is assigned to machine k. We compute the heuristic incremental cost for every machine k, and assign product i to the machine with the lowest incremental cost.

The final assignment of products to machines suggested by the list processing heuristic might violate the workload constraint (1.4). If it does, we attempt to transfer products from overloaded machines to others that have slack capacity. From among the products assigned to the most overloaded machine, we select the product whose removal gives the maximum reduction in setup cost on that machine, and can be feasibly accommodated on another machine, i.e., the processing time for this product must be no more than the slack time

available on at least one other machine. Among the machines that have adequate slack time, we choose the machine that incurs the lowest (heuristic) incremental setup cost, and reassign the product to this machine. Because this strategy is myopic, it might terminate without producing a feasible product-to-machine assignment.

3.3.2 Smallest setup–to-processing ratio heuristic

This method is better suited than the previous list processing heuristic when the workload balancing constraints are tight. The procedure to select the first product for each machine is the same as before. Subsequently, the algorithm iteratively selects the least loaded machine k, and assigns to it the unassigned product i that minimizes the ratio δ_{ik}/P_i^{β} of the incremental setup cost to processing time, where δ_{ik} is the incremental setup cost if we add product i to machine k, and the exponent β is a user-specified parameter. Again, we apply the SO heuristic to approximate the incremental setup cost at each step.

Initially, we set $\beta = 0$. If the final product-to-machine assignment violates the workload balancing constraints, we increase the value of β and repeat the algorithm. As β increases, the algorithm resembles the Longest Processing Time (LPT) heuristic which has proven effective for minimizing the makespan for parallel machines (see, for example, Frenk and Rinnooy Kan [1984]). For all of our test problems, the processing limit T exceeds the LPT makespan; hence, this setup-to-processing ratio heuristic always generates a feasible heuristic solution.

We select the better (i.e., having lower heuristic setup cost) of the list processing and smallest ratio heuristic solutions as the initial solution for the tactical planning model, and evaluate its optimal setup cost by solving the corresponding SO problems optimally. We can develop many other heuristic procedures for the product assignment problem such as the group technology methods that we outlined in Section 2, or analogs of bin packing and parallel machine scheduling algorithms. We have not pursued these alternate heuristic strategies since we use the initial heuristic solution mainly as a starting point for the column generation algorithm.

3.4 The Column Generation Algorithm

Having constructed an initial heuristic solution, Phase 2 attempts to solve the LP relaxation [LCSC] of the capacitated set covering formulation [CSC] in order to generate lower bounds on the optimal setup cost Z^*, and possibly improve the feasible solution. Recall that the H columns of formulation [CSC] correspond to all the assignment patterns with total processing time less than or equal to the processing limit T. The h^{th} pattern

consists of assigning all products with $a_{ih} = 1$ to a single machine; its cost coefficient $s^*(\bar{a}_h)$ in [CSC] is the minimum setup cost, obtained by solving the corresponding SO problem optimally.

Starting with the K assignment patterns of the initial heuristic solution as the initial set of columns, the column generation procedure iteratively generates new "promising" columns as needed until the linear programming optimality conditions of [LCSC] are satisfied. Let H^t denote the number of assignment patterns or columns available at the start of iteration t of the column generation procedure. Initially, $H^0 = K$, the initial heuristic solution is the current incumbent, and the current best upper bound is the total setup cost of this solution. At iteration t, we first solve the following *restricted master problem*, denoted as [RM^t], containing the H^t columns generated thus far:

[RM^t]

$$Z_{LP}^t = \min \sum_{h=1}^{H^t} s^*(\bar{a}_h)\,\lambda_h \qquad (3.26)$$

subject to

$$\sum_{h=1}^{H^t} a_{ih}\,\lambda_h \geq 1 \qquad \text{for all } i = 1,2,\ldots,N, \quad (3.27)$$

$$\sum_{h=1}^{H^t} \lambda_h \leq K, \text{ and} \qquad (3.28)$$

$$\lambda_h \geq 0 \qquad \text{for all } h = 1,2,..,H^t. \quad (3.29)$$

The optimal value Z_{LP}^t is an upper bound on Z_{LP} since the restricted master problem contains only a subset of the columns in the original LP relaxation [LCSC]. Let $\lambda^t \in \mathbf{R}^{H^t}$ be the optimal solution to [RM^t], and let π_i^t and σ^t denote the optimal dual values corresponding to constraints (3.27) and (3.28) respectively.

If the LP solution λ^t is integral, then it is feasible for the original problem (it need not be optimal since [RM^t] does not contain all of [LCSC]'s columns). If this *intermediate solution* has lower total setup cost than the current best upper bound, we update the current incumbent and upper bound.

To verify if the solution λ^t is optimal to [LCSC] we must check if any other column of [LCSC] (not in [RM^t]) has negative reduced cost using the optimal dual values π_i^t and σ^t. If not, we have solved [LCSC] optimally, and $Z_{LP} = Z_{LP}^t$ is a valid lower bound on the optimal

value Z^* of the tactical planning problem. Otherwise, we generate one or more columns with negative reduced cost, add these columns to the restricted master problem, and perform the next iteration. To check for optimality and generate negative reduced cost columns at each iteration, we solve the *product selection* subproblem.

3.4.1 The product selection subproblem

At iteration t, the reduced cost of any column or assignment pattern $\bar{a}$ with optimal setup cost $s^*(\bar{a})$ is

$$\rho(\bar{a}) \quad = \quad s^*(\bar{a}) - \sum_{i=1}^{N} a_i \pi_i^t + \sigma^t. \tag{3.30}$$

Note that every column in the current restricted master $[RM^t]$ must have nonnegative reduced cost since π_i^t and σ^t are the optimal dual values for $[RM^t]$. To verify if any other column of [LCSC] has negative reduced cost, we attempt to find a feasible assignment of products to a single machine that minimizes (3.30). We can formulate this minimization problem as the following *product selection* subproblem $[PS^t]$ using binary variables similar to those in formulation [P], but for a single machine. The variables of $[PS^t]$ correspond to product selection (x_i), permanent component loading (y_j), and temporary component loading (z_{ij}) decisions:

$$[\mathbf{PS}(\pi^t)] \qquad L(\pi^t) \quad = \quad \min \quad \sum_{i \in I} b_i \{ \sum_{j \in J(i)} z_{ij} \} - \sum_{i=1}^{N} \pi_i^t x_i \tag{3.31}$$

subject to

Maximum workload constraint:

$$\sum_{i=1}^{N} P_i x_i \quad \leq \quad T\,, \tag{3.32}$$

Component loading constraints:

$$y_j + z_{ij} \quad \geq \quad x_i \quad \text{for all } i = 1,2, \ldots,N, \text{ all } j \in J(i), \tag{3.33}$$

Slot capacity constraints:

$$\sum_{j \in J} y_j + \sum_{j \in J(i)} z_{ij} \quad \leq \quad B \quad \text{for all } i = 1,2, \ldots,N, \text{ and} \tag{3.34}$$

Integrality constraints:

$$x_i, y_j, z_{ij} \quad = \quad 0 \text{ or } 1 \quad \text{for all } i = 1,2, \ldots,N, \text{ all } j \in J(i). \tag{3.35}$$

We interpret π_i^t as the reward or profit for assembling product i. We must decide which products to accept ($x_i = 1$ if we accept product i, and 0 otherwise) in order to maximize the "net profit", which is the total profit for all accepted products minus the total setup cost. Having selected the products, we must also decide which of the required components to load permanently; the remaining components are temporary, and we incur their setup costs for each chosen product that uses these components.

Since each column in the optimal basis of $[RM^t]$ has zero reduced cost, and is feasible for $[PS(\pi^t)]$, the optimal PS subproblem value $L(\pi^t)$ must be less than or equal to $-\sigma^t$. Note that we can proceed with the next iteration of the master problem as long as we generate some negative reduced cost column, not necessarily the optimal solution to $[PS(\pi^t)]$. Since solving $[PS(\pi^t)]$ optimally using a general-purpose branch-and-bound method is very time consuming (the LP lower bound for formulation $PS(\pi^t)$ is weak), we first apply heuristic methods to find one or more product assignment patterns with negative reduced cost (although we select the products heuristically, we compute the pattern's true setup cost by solving the corresponding SO problem optimally). If this effort is unsuccessful (i.e., all heuristic patterns that we generate have nonnegative reduced cost), we must solve $[PS(\pi^t)]$ optimally. If the branch-and-bound procedures generates a pattern with negative reduced cost (at an intermediate or final stage), we can add the corresponding column to the restricted master problem, and initiate the next iteration. Otherwise, if $L(\pi^t) = -\sigma^t$, the current LP solution λ^t is optimal for [LCSC], and the current optimal value of the restricted master problem is a valid lower bound on Z^*.

Before describing our heuristic method to solve the product selection subproblem, let us discuss some strategies that might improve the performance of the column generation procedure by reducing the number master of iterations needed to converge to the optimal solution. First, although generating just one negative reduced cost column is sufficient to initiate the next iteration of the master problem, Desrochers et al. [1992] have noted that generating multiple columns at each iteration can accelerate convergence. Furthermore, they recommend generating disjoint columns to improve the chances of obtaining an integer optimal solution to the restricted master problem. In our problem context, disjoint columns correspond to feasible single machine assignments of mutually exclusive product subsets.

A second improvement strategy consists of adapting Magnanti and Wong's [1984] acceleration technique for Benders' decomposition to the column generation algorithm. When the Benders' subproblem has alternate optima, the method selects pareto optimal solutions by optimizing a subsidiary objective. This pareto optimality principle also extends to our column generation procedure; however, it requires optimally solving the difficult

product selection subproblem twice. Instead, we use insights from the principle to modify our heuristic procedure. In particular, Magnanti and Wong's pareto optimality result implies that among the alternate optima for the product selection subproblem [PS(π^t)], we prefer: (i) a pattern that assigns more products to the machine, with ties resolved in favor of the pattern with the lowest setup cost, or (ii) a pattern that has the smallest possible setup cost, with preference for a pattern that assigns more products to the machine (see Vanderbeck [1993]). We incorporate these preferences in the heuristic column generator that we describe next.

Product Selection (PS) Heuristic

For any feasible board assignment pattern satisfying (3.32), we will refer to its optimal setup cost minus the total profit for all the chosen products as the *net setup cost.* Adding σ^t to the net setup cost of an assignment pattern gives its *reduced cost.* To compute the reduced cost of an assignment pattern, we must evaluate its true optimal setup cost by solving the corresponding SO problem optimally. However, an approximate SO method (that provides an upper bound on the setup cost) might be adequate for evaluating alternative assignment patterns during the heuristic search process. If, for a particular assignment pattern, the upper bound on net setup cost obtained using the approximate SO method is less than $-\sigma^t$, then the true reduced cost for this pattern must be negative, and so we can add the corresponding column to the restricted master problem (after evaluating its true reduced cost).

The PS heuristic seeks to minimize net setup cost. The method has two components: a sequential selection heuristic to first select a subset of products that satisfies the maximum workload constraint, and a neighbor scanning procedure to further decrease the reduced cost of this heuristic solution or generate other negative reduced cost columns. The **Sequential Selection heuristic** successively selects products that provide the maximum incremental benefit. At any step l of the procedure, let I_l denote the subset of $(l-1)$ products that the method has chosen in the previous $(l-1)$ steps. For every remaining product i whose processing requirement is less than the current slack capacity, we evaluate the incremental net profit which is π_i minus the incremental setup cost if we add product i to I_l. As in the initial heuristic procedure, we estimate the incremental setup cost by applying the SO heuristic. Let i* be the product with the maximum (heuristic) incremental net profit. If product i*'s incremental net profit is negative, we terminate the procedure; otherwise, we add product i* to I_l and repeat the procedure until no remaining product has processing requirement less than or equal to the residual time on the machine. We attempt to satisfy the pareto optimality conditions by: (i) selecting product i* even if it has zero marginal benefit (in order to maximize the number of products assigned to the machine), and (ii) selecting the product with the lowest incremental setup cost when two or more products have the same marginal benefit.

When the sequential selection heuristic terminates, we evaluate the true setup cost of the chosen products by solving the SO problem optimally. If the reduced cost is negative, we add the corresponding column to the restricted master problem. The second phase of the PS heuristic consists of applying a **neighbor scanning procedure** to the sequential heuristic solution; we apply this procedure even if the sequential heuristic solution has negative reduced cost. Starting with the sequential heuristic solution as the initial incumbent, the neighbor scanning procedure estimates (using the SO heuristic) the reduced cost for all the neighbors of the incumbent. We define the neighborhood of an incumbent via 1-exchange and 2-exchange operations that remove one or two currently chosen products and add upto 3 new products. If the procedure finds any promising pattern with an estimated reduced cost that is negative or lower than the incumbent's reduced cost, it scans the neighborhood of this pattern through a recursive call to itself.

Consider an incumbent pattern whose neighborhood we want to search. Let I' denote the set of products chosen in this pattern, and let I'' $\subseteq$ N' be the current set of candidate products. Initially I' is the subset of products chosen by the sequential heuristic, and I'' = N'; we delete products from the candidate set I'' at selected intermediate steps of the neighbor scanning procedure in order to generate disjoint columns. The neighbor scanning procedure performs the following steps:

Neighbor Scanning Procedure

Stage 1: 1-exchange:

For each product $i \in I'$, and every pair of products $j,k \in I''$,

1A: check if the solutions $I' \setminus \{i\}$, $I' \setminus \{i\} \cup \{j\}$, $I' \setminus \{i\} \cup \{k\}$, and $I' \setminus \{i\} \cup \{j,k\}$ satisfy the maximum workload constraint (3.32);

1B: evaluate the heuristic reduced cost of each new pattern;

1C: if a new pattern has a negative or lower heuristic reduced cost than the incumbent, evaluate its exact setup and reduced cost;

1D: if the exact reduced cost is negative, add the corresponding column to the master problem, and remove the products chosen by that pattern from the set of candidate products I''.

1E: if the exact reduced cost is negative or lower than the incumbent's reduced cost, call the Neighbor Scanning Procedure with the new pattern as incumbent;

If none of the new patterns generated by 1-exchange have negative or lower reduced cost than the incumbent pattern, go to Stage 2.

Stage 2: 2-exchange:

For every pair of products $i, j \in I'$, and every triplet of available products $u, v, w \in I''$,

2A: check if the solutions I\{i,j}, I\{i,j}∪{u}, I\{i.j}∪{v}, I\{i.j}∪{w}, I\{i.j}∪{u,v}, I\{i.j}∪{v,w}, I\{i.j}∪{u,w}and I\{i,j}∪{u,v,w} satisfy the maximum workload constraint (3.32);

Apply Steps 1B, 1C, 1D, and 1E to each new pattern.

The neighbor scanning procedure recursively searches the neighborhood whenever it generates an improved product assignment pattern with lower or negative reduced cost (see Step 1E) since this pattern is likely to have promising neighbors. Therefore, the procedure can potentially generate numerous negative reduced cost columns at each iteration, thus decreasing the number of master iterations, reducing the number of times we need to solve the PS subproblem optimally, and also providing a wider choice of promising columns at the final iteration. In order to limit the search process, we impose an upper limit called the *maximum depth* of search on the number of recursive calls.

If, at a particular master iteration, the PS heuristic fails to generate a negative reduced cost column, we must solve the PS subproblem optimally for the current set of optimal dual values π^t and σ^t. We use branch-and-bound to solve [PS(π^t)] optimally; since the columns in the current optimal basis have zero reduced cost and are feasible for [PS(π^t)], we specify $-\sigma^t$ as the a priori upper bound for the LP-based branch-and-bound procedure. Our implementation uses a standard branch-and-bound code (CPLEX), and therefore does not incorporate the iterative upper and lower bounding methods (described in Section 3.2) for the setup optimization subproblems at each node of the branch-and-bound tree. The code permits us to assign branching priorities to different variables; we branch first on the product assignment variables x_i, next on an auxiliary variable W for the number of permanent components, and finally on the permanent component loading variables y_j.

If the PS heuristic or the subsequent branch-and-bound algorithm produces a negative reduced cost column(s), we attempt to generate other disjoint negative reduced cost columns by deleting from the product list all the products chosen previously by the sequential selection heuristic or the optimal PS solution, and reapplying the PS heuristic with the reduced set of products.

3.5 Intermediate Lower Bounds on Z^*

Optimally solving the LP relaxation [LCSC] of the capacitated set covering formulation gives the lower bound Z_{LP} on the optimal value Z^* of the original tactical planning problem. In the column generation scheme, the value Z_{LP}^t of the restricted master problem at an

intermediate iteration t is not a valid lower bound on Z^*; we get the LP lower bound Z_{LP} only at the final master iteration when we have proved that no other single machine assignment pattern has negative reduced cost. As we explain in Section 4, our implementation limits the number of nodes in the branch-and-bound tree, and so might terminate without solving the PS subproblem, and hence [LCSC], optimally. However, as the following discussion shows, we can use the optimal values (or lower bounds on these values) of the product selection subproblem at intermediate iterations to develop lower bounds on Z^*.

Consider the original formulation [P] of the tactical planning problem, and suppose we dualize the product assignment constraints (1.2) using Lagrangian multipliers π_i for all i = 1,2,...,N. Then, the problem decomposes into K subproblems, one for each machine. For machine k, the Lagrangian subproblem seeks a subset of products to assemble on this machine that minimizes the net setup cost subject to the workload balancing constraint (1.3), component loading constraints (1.4), and slot capacity constraints (1.5) for machine k. If all the machines are identical, i.e., $P_{ik} = P_i$, $T_k = T$, and $C_k = C$ for all k =1,2,...,K, then the same product assignment is optimal for all K Lagrangian subproblems. Observe that the single-machine Lagrangian subproblem is the same as the product selection subproblem [PS(π)] of the column generation scheme. Therefore, if we use the optimal dual values π_i^t of the restricted master problem [RM^t] as the Lagrangian multipliers, we get the following valid Lagrangian-based lower bound

$$Z_{LB}^t = \sum_{i=1}^{N} \pi_i^t + K\, L(\pi^t), \qquad (3.36)$$

on Z_{LP} at every iteration t of the master problem. Assuming that all the production frequency parameters b_i are integers, we can round Z_{LB}^t up to get a valid lower bound on Z^*. In our column generation scheme, we can compute this lower bound whenever we solve the PS subproblem optimally (recall that we solve the PS subproblem optimally only when the PS heuristic cannot identify any negative reduced cost column). Note, however, that if we could generate a lower bound on the optimal value $L(\pi^t)$ of the PS subproblem (using, say, a Lagrangian relaxation method for this subproblem), then we can substitute this lower bound for $L(\pi^t)$ in (3.36) to get a valid lower bound on Z_{LB}^t, and hence Z_{LP}.

We update the current best lower bound if the new integer lower bound is higher, i.e.,

$$\text{Best lower bound at iter. } t = \max \{\text{Best lower bound at iter. } (t-1), \lceil Z_{LB}^t \rceil\}. \qquad (3.37)$$

If this integer lower bound exceeds the current optimal value of the restricted master problem, then we can terminate the column generation procedure since pursuing the optimal

solution of [LCSC] will not produce a tighter lower bound on Z^*. Thus, by rounding up Z_{LB}^t, we can potentially eliminate several master iterations at the end. This strategy can provide considerable computational savings since the optimal value of the master problem tends to converge very slowly towards the end, and the final iterations are computationally intensive (since they frequently require solving the PS subproblem optimally using branch-and-bound). Finally, although Z_{LB}^t is less than the optimal value Z_{LP}^t of the master problem [RM^t] at intermediate iterations t of the column generation procedure, at the final iteration t_f (assuming we pursue the column generation procedure until the optimal solution to the restricted master problem solves [LCSC]) the lower bound $Z_{LB}^{t_f}$ equals Z_{LP} since $L(\pi^{t_f}) = -\sigma^{t_f}$. Note also that Z_{LP}^t decreases monotonically, whereas Z_{LB}^t can increase or decrease at each iteration.

3.6 Summary

Figure 1 summarizes our solution procedure for the tactical planning model. We begin by generating an initial heuristic solution using the greedy heuristic, with the setup optimization method as a subroutine. This solution provides the first K columns of the master problem for the column generation algorithm. Each iteration of the column generation method entails solving the product selection subproblem using the optimal dual values for the current master iteration. We first attempt to generate negative reduced cost columns heuristically using the sequential selection and neighbor scanning procedures; this PS heuristic also uses the setup optimization method (approximate and optimal) as a subroutine. If we cannot find any single-machine product assignment with negative reduced cost, we apply branch-and-bound to solve the PS subproblem optimally. In this case, we also compute the Lagrangian-based lower bound (equation (3.36)), round it up, and update the current best lower bound as in (3.37). Also, whenever we add a negative reduced cost column to the restricted master problem, we remove the products chosen by this column and reapply the PS heuristic in order to generate disjoint columns.

If the optimal LP solution to the restricted master problem at an intermediate iteration is integral, we obtain an intermediate feasible solution. We update the current incumbent if this solution has lower total setup cost. The column generation procedure terminates when the best lower bound equals or exceeds the current optimal value of the restricted master problem. We then construct a **final feasible solution** by solving the restricted master problem as an integer program. The solution to this integer program is not necessarily optimal to [CSC] since the restricted master problem does not contain all of [CSC]'s columns. On the other hand, since the final set of columns includes the product assignments

belonging to the initial and intermediate feasible solutions, the final solution must have equal or lower setup cost than the current incumbent.

The setup optimization subroutine incorporates heuristic procedures to develop good component loading plans as well as a method to compute a lower bound on the optimal setup cost. The lower bound is based on an underestimate of the minimum number of temporary slots necessary to ensure feasibility of the slot capacity constraints. We also add a valid inequality limiting the maximum number of permanent components that strengthens the LP relaxation of the SO problem formulation. To solve the setup optimization problem optimally (e.g., to estimate the true reduced cost of candidate columns), we use branch-and-bound (unless the lower bound equals the setup cost of the heuristic solution); the heuristic upper and lower bounds serve to limit the enumeration process.

Finally, note that all the methods that we have described extend easily to nonhomogenous machines with different processing speeds, processing limits, and slot capacities. For instance, when the machines are not identical, each machine has its own set of columns in formulation [CSC] and in the restricted master [RM^t]; instead of a single pattern selection constraint (3.7), we now have K constraints (3.3) (with corresponding dual values σ_k^t) specifying that we can select at most one pattern for each machine. At every iteration we must solve K separate PS subproblems, and the procedure terminates only when none of the machines has an assignment pattern with negative reduced cost, or the best lower bound exceeds the optimal value of the restricted master problem.

4. Computational Experiments

In implementing solution methods such as column generation, we are faced with a variety of tradeoffs and algorithmic options that can impact the performance of the algorithm. These tradeoffs raise issues such as the following: What is the impact of initializing the procedure with columns corresponding to a good heuristic solution versus, say, randomly generated columns? Should we invest more effort in heuristically generating negative reduced cost columns or is it better to solve the subproblem optimally at each iteration? To address some of these issues and verify the column generation method's viability and effectiveness for the electronics assembly tactical planning context, we implemented the method and performed several computational experiments. We note that unlike many column generation applications discussed in the literature that have easy subproblems, for the tactical planning model even the core setup optimization subproblem is NP-hard.

We implemented the heuristics and column generation procedure using the C programming language on a Sun SPARC workstation. The program calls subroutines from the CPLEX linear optimizer (CPLEX Optimization Inc. [1991]) to solve the LP relaxation of the restricted master problem at each iteration, and the product selection and setup optimization problems whenever we need to solve these subproblems optimally. To solve these latter integer programs, our implementation only provides an initial upper bound, and sets the branching priorities for different variables; we rely on CPLEX's general purpose LP-based procedure to generate lower bounds during the branch-and-bound process.

4.1 Test Problems

The tactical planning model requires the following input data (assuming identical machines):

N: number of products;
M: total number of component types required to assemble the N products;
K: number of available machines;
C: slot capacity of each machine;
T: processing limit of each machine;
b_i: expected number of batches per demand period for product i;
P_i: total processing (placement) time to assemble all required units of product i; and,
J(i): set of component types required for product i.

Table 1 shows the problem size and other parameters for 20 test problems. The number of products varies from 9 to 20, and the number of components from 18 to 60. The problem name specifies the number of products and component types; we use the trailing letter to distinguish between multiple instances of the same problem size (these instances differ in other problem parameters). The detailed data for the problems with 9 products, 10 products, and 20 products, 20 components are given in Vanderbeck [1993]. Of the 20 problems, we generated the first 5 problems manually in order to study the impact of different bill of components structures. In problems P10/18A and P10/18B, the 10 products form 3 groups with modest overlap in the component requirements for each group; in contrast, problems P09/18A and P09/18B have 3 groups of products with no component overlap. Finally, problem P09/18C has a block diagonal bill of components matrix.

To generate larger problems (with 15 and 20 products), we used a random problem generator that constructs a detailed problem instance based upon user-specified values for the desired problem size, a seed for the random number generator, and certain control parameters that determine the structure of the bill of components. The problem size

parameters are N, M, K, and C. To influence the degree of component commonality among different products, the user specifies two parameters–M_{min} and M_{max}–that define the minimum and maximum number of component types required for each product. For every product i, the problem generator first randomly selects the number of components m_i from the interval $[M_{min},M_{max}]$, and then chooses m_i component indices randomly (without replacement) from the set {1,2,...,M}; these component types define the set J(i). Note that the slot capacity C must be greater than or equal to M_{max} in order to ensure that every product can be assembled in a single pass on a machine. By setting M_{min} and M_{max} close to the total number of component types M, we can generate problem instances that have high commonality; in particular, if $M_{min} = M_{max} = M \leq C$, then all components can be permanently loaded on each machine, and the optimal value Z^* is zero. At the other extreme, if M_{min} and M_{max} are very small relative to M, then the products are likely to have low component commonality. Specifying small M_{min} and large M_{max} will likely increase problem difficulty since the bill of components for different products do not have any special commonality structure.

The problem generator randomly selects the processing times P_i and the number of batches b_i from uniform distributions: $P_i \sim$ Unif (5,20), and $b_i \sim$ Unif $(1,P_i)$. We then apply the LPT (longest processing time) heuristic to approximate the minimum makespan. The LPT heuristic (Baker [1974]) sorts the products in decreasing order of P_i, and assigns each product in sequence to the first available machine. This heuristic produces a product-to-machine assignment with a makespan T_{LPT} that is at most (4/3 – 1/3K) times the minimum makespan (Graham [1969]). For our test problems, we set the processing limit T equal to α T_{LPT}, where α (≥ 1) is a user-specified *tolerance factor*. This method of choosing T ensures that each problem instance has at least one feasible solution satisfying the workload balancing constraints. For most of our test runs, we set $\alpha = 1.2$, although we also experimented with problem instances having "tight" ($\alpha = 1.05$) and "loose" ($\alpha = 1.5$) processing limits.

The random problems P20/20A to P20/20E enable us to assess the sensitivity of the algorithm's performance to slot capacity, and the level of variation in the number of component types per board. We also use these 5 problems to test several algorithmic variants. Problems 15/20 to P20/60 enable us to evaluate the impact of problem size on performance.

4.2 Variants of the Solution Procedure

Using a relatively flexible implementation, we were able to test several variants of the tactical planning algorithm described in Section 3. These variants include (i) generating only one negative reduced cost column at each master iteration, (ii) discarding all the current columns with positive reduced cost at each master iteration, (iii) varying the tolerance factor α, (iv) changing the depth of search and the criterion for recursive calls in the neighbor scanning procedure of the PS heuristic, and (v) reapplying the column generation procedure using the columns of the best feasible solution (obtained after applying a standard version of the algorithm) as the initial set of columns.

Recall that the PS heuristic, as we described it in Section 3, first applies the sequential search heuristic, and then invokes the neighbor scanning procedure. The neighbor scanning procedure recursively explores the neighborhood of each generated pattern that has negative reduced cost or a lower reduced cost than its predecessor. Our initial implementation, which we call the **basic** version, does not incorporate the neighbor scanning procedure in the PS heuristic. Our **standard** version performs neighbor scanning but limits the number of recursive calls to at most 2. We also performed computational tests using different values for the **maximum depth of search,** and tested a variant of the neighbor scanning procedure we call the **reduced cost updating** version. If the current incumbent has nonnegative reduced cost, the *reduced cost updating* version explores the incumbent's neighborhood only if its reduced cost is lower than the current lowest reduced cost (rather than the reduced cost of its predecessor, as in the standard version; however, both versions explore the neighborhood of every incumbent with negative reduced cost). Therefore, the reduced cost updating version explores much fewer neighbors than the standard version.

Our standard version generates (if possible) multiple negative reduced cost columns at each master iteration, and retains all the columns in the restricted master. We tested two variants of this column generation strategy–a **single column** version, which generates only one column per master iteration, and a **condensed master** version, which limits the size of the master problem by discarding all the columns with positive reduced cost, i.e., we only include in the restricted master formulation those columns belonging to the current optimal basis plus prior and new columns that price out negatively using the current dual values. Section 4.3.2 presents the comparative results for all these algorithmic variants, as well as other sensitivity analyses.

We now discuss some features of our implementation that is common to all the variants. If the PS heuristic fails to produce any negative reduced cost column, we solve the PS

subproblem exactly using CPLEX's built-in branch-and-bound procedure. For our computational tests, we specified a limit of 10,000 nodes for the branch-and-bound tree. We also terminate column generation if it "stagnates", i.e., if it does not produce any improvement in the restricted master value for 500 consecutive master iterations or after optimally solving the PS subproblem 2N times. The PS solution procedure can, therefore, have four outcomes: (i) the branch and bound procedure reaches its node limit without finding a negative reduced cost column, (ii) we generate a negative reduced cost column, but the node limit is reached before finding and/or verifying the optimal solution, (iii) we solve the PS subproblem optimally and find a negative reduced cost column, or (iv) the optimal solution to the PS subproblem has zero reduced cost. In case (i), the column generation procedure stops since we did not generate a new column for the restricted master. In this case and also when column generation terminates due to stagnation, the optimal LP value of the current master problem is not a valid lower bound on Z^* and so we must rely on the best lower bound obtained during previous iterations to assess the quality of the heurisic solution. In cases (ii) and (iii) we add the new negative reduced cost column to the restricted master, and continue the iterations. Case (iv) proves the optimality of the current master solution for [LCSC], and so column generation terminates.

To generate the final feasible solution (in Phase 3), we initially implemented a *simulation annealing* algorithm to select a good subset of K assignments from the set of all columns in the restricted master problem at the final iteration. However, we found that optimally solving the integer version of the restricted master using branch-and-bound is relatively easy since the LP relaxation provides tight bounds and almost integral solutions (because we generate several disjoint columns at each master iteration). We, therefore, applied this method to all the test problems. We expect the quality of the final feasible solution to improve as the number of columns available at the final iteration increases.

4.3 Computational Results

We first present the computational results using the *standard* version of the column generation procedure for our 20 test problems before discussing the relative performance of some variants. We are interested in assessing both the algorithm's *effectiveness* in generating near-optimal solutions, and the *computational effort* it requires. We measure effectiveness in terms of the *% gap*, which we define as the difference between the best upper and lower bounds, expressed as a proportion of the best lower bound. The % gap consists of three components:

(i) the gap between the optimal value of [LCSC] and the best lower bound (computed using (3.36) and (3.37)), which we call the *dual optimality gap*. This gap is positive

when the column generation procedure terminates prematurely due to the branch-and-bound node limit or stagnation of the restricted master value. If the procedure terminates because the best lower bound exceeds the current value of the restricted master problem, then the dual optimality gap is negative;

(ii) the gap between the optimal value Z^* and the optimal value Z_{LP} of the LP relaxation [LCSC], which we call the *duality gap*; and

(iii) the gap between Z^* and the setup cost of the final feasible solution, which we call the *primal optimality gap*.

Thus, small overall gaps indicate that both the upper and lower bounding methods are effective. To determine if the column generation-based feasible solutions significantly improve upon the initial heuristic solution, we examine the improvement in setup cost (i.e., the difference in the setup costs of the initial and best final solutions) as a % of the initial solution's setup cost.

To measure computational effort and identify bottlenecks, we follow Ahuja, Magnanti, and Orlin's [1993] suggestion to use computation counts (e.g., number of iterations) for major segments of the algorithm rather than actual CPU time which can be misleading since it depends on the programming language, quality of implementation, computer system used, and other characteristics of the computing environment. For each test problem we examine the following statistics (see Table 2):

(i) number of columns generated for the master probem,

(ii) number of iterations of the master problem,

(iii) number of times we solved the PS subproblem exactly using branch-and-bound,

(iii) number of times we solved the SO subproblem exactly using branch-and-bound,

(iv) percentage of instances when the SO heuristic produced the optimal slot allocation and proved optimality (i.e., when heuristic cost was equal to the lower bound), and

(v) number of nodes in the branch-and-bound tree to solve the integer version of the restricted master problem at the final step.

4.3.1 Results for standard version

Table 2 shows the results for the 20 test problems using the standard version of our algorithm. The first column (labeled "[LCSC] Solved?") shows whether the procedure was able to solve the LP relaxation [LCSC] optimally.

For 4 out of the 20 problems the algorithm terminated prematurely because the branch-and-bound procedure for the PS subproblem reached its node limit before finding a negative reduced cost column or verifying optimality. The % gap is less than 4% for 3 out of these 4 problems. Recall that, when the column generation procedure terminates prematurely, we

measure the % gap relative to the best lower bound that is computed at intermediate iterations of the column generation procedure (whenever the PS subproblem is solved optimally). Therefore, the actual gap between the optimal value of [LCSC] and the final feasible solution is likely to be lower. We suspect that, for Problem P20/40A, the final feasible solution is much closer to optimality than the 20% gap suggests.

The results in Table 2 suggest that the column generation approach is very effective. It solved 10 out of the 20 test problems optimally, and excluding Problem P20/40A the average % gap for the remaining problems (with positive gap) is 4.6 %. This result indicates that the LP relaxation of [CSC] is tight, and the procedure is able to generate near-optimal product-to-machine assignments. The % gap does not show any systematic variation with the slot capacity, the range of number of component types per board, or problem size. For all but one test problem, the best column generation-based feasible solution, obtained as a byproduct of solving the restricted master problem, has lower total setup cost compared to the initial heuristic solution obtained using the list processing and smallest ratio heuristics. On average, the final solution had 22 % lower cost than the initial heuristic solution (admittedly, we can develop ad hoc heuristics that improve upon the initial product assignment heuristics that we applied). As we note later when we report the results for algorithmic variants, generating more columns at each iteration produces better intermediate solutions. The LP solution to the restricted master problem in the final iteration is often integral (in 12 out of the 20 problem instances); if not, the branch-and-bound procedure solves the integer program within less than 100 nodes.

For all the test problems we observed that solving the PS subproblem exactly is the main computational bottleneck, requiring one or two orders of magnitude more time than the other components of the algorithm (this assessment is based on a rough sampling of the elapsed times; we did not record the actual computation times). In contrast, the SO problem appears to be easy to solve. Although Table 2 shows that the % of SO problems requiring branch-and-bound increases as the problem size increases, we observed that the SO heuristic actually generates the optimal solution in over 85% of the instances; however, the lower bound (based on the underestimate of the minimum number of temporary components) is not tight enough to prove optimality of the heuristic solution. When we apply branch-and-bound to verify optimality, the branch-and-bound tree for the SO problem has 0 nodes in over 90% of the cases. These observations suggest that the total computation effort will likely be more sensitive to the number of products compared to the number of component types. Finally, during the initial iterations of the master problem the PS heuristic readily find numerous negative reduced cost columns. Indeed, a vast majority of the columns are quickly generated in these initial iterations. Although the algorithm finds good assignment patterns early in the

iterative column generation process, proving the near-optimality of the feasible solution is very time consuming especially since the later iterations require solving the PS subproblem optimally using branch-and-bound.

4.3.2 Results for algorithmic variants

Table 3 contains the computational results using several variants for selected test problems. For each test problem, we indicate with a "**" the method that is most effective in terms of % gap; if two methods produce the same % gap, we prefer the method that entails fewer applications of the branch-and-bound procedure to solve the PS subproblems optimally.

The *basic* version, which does not incorporate the neighbor scanning procedure in its PS heuristic, applies branch-and-bound more often to solve the PS subproblems, generates fewer columns at each iteration thus requiring more master iterations to converge, and often produces inferior final feasible solutions. Similarly, the *single column* version is not as effective as the standard version. Our third variant, the *condensed master* version, has the advantage of reducing the time to solve the restricted master problem, but greatly increases the number of master iterations. Interestingly, although the procedure often terminates prematurely (because the restricted master value stagnates), it generates many more columns, thus producing better final solutions (we include all the columns before solving the integer version of the restricted master problem at the final step).

The fourth and fifth versions reported in Table 3 represent problem instances with tight and loose processing limits (with tolerance factor α = 1.05 and 1.5) to which we apply the basic version. Tightening the processing limit reduces the feasible solution set, but also limits the flexibility in optimizing setups; these opposing effects reduce the % gap for some problems, but increase the gap for others. We also experimented with different settings for the maximum depth of search–unrestricted[8], maximum depth of 3, and maximum depth of 2–in the neighbor scanning procedure. With a maximum depth of 3, we also considered results with and without the "reduced cost updating" option (i.e., the criterion for initiating a recursive call to the neighborhood search subroutine, see Section 4.2). The unrestricted depth and depth 3 versions generate more columns than the standard depth 2 version, and require solving considerably more setup optimization subproblems, but do not produce superior solutions (i.e., the % gap is often equal or lower for the depth 2 version). Finally, to study the impact of selecting good initial columns for the restricted master problem, we

[8] In the "unrestricted" depth of search option, we do not impose an explicit limit on the number of recursive calls to the exchange heuristic. However, because we only search the neighborhood if the incumbent's reduced cost is negative or improves upon its predecessor's, the actual depth of search is naturally limited.

compared the results using our standard version (which uses the assignments produced by the better of our two initial greedy heuristics as the initial set of columns) with those of a second application of the solution procedure when we use the best feasible solution from the standard version to initialize the restricted master. This strategy does not produce lower % gaps than the other methods, and can sometimes require considerably more computational effort.

4.4 Potential Algorithmic Improvements

Our computational experience suggests that efforts to improve the capabilities and performance of the algorithm should mainly focus on solving the single machine product selection problem more effectively. In particular, because the linear programming relaxation of formulation [PS(π)] is relatively weak, we must incorporate special bounding procedures and attempt to strengthen the problem formulation by adding valid inequalities that eliminate fractional solutions.

As we noted previously, our implementation relies on CPLEX's built-in branch-and-bound procedure to solve the PS subproblem optimally, and therefore does not exploit the special structure of this subproblem. Implementing a custom branch-and-bound procedure to incorporate valid inequalities and/or problem-specific bounding procedures at each node has the potential to substantially improve performance. Let us first introduce some notation before describing some inequalities that strengthen the formulation. At any node of the branch-and-bound tree, let I_1, I_0, and I_f denote, respectively, the subsets of products that are already chosen, eliminated, and free, i.e., $x_i = 1$ for all $i \in I_1$, and $x_i = 0$ for all $i \in I_0$.

(i) Using the method described in Section 3.2.4, we can compute an upper bound $R(I_1)$ on the maximum possible number of permanent slots in order to ensure feasibility of the slot capacity constraints for all products in I_1. Since selecting more products from I_f can only decrease this value, we can add the valid inequality that the number of permanent slots chosen in the current partition must not exceed $R(I_1)$. We have observed that this inequality markedly improves the performance of the setup optimization branch-and-bound procedure; we expect it to be equally effective for the PS subproblem when I_f contains relatively few products.

(ii) Pochet and Wolsey [1992] suggested the following class of valid inequalities to strengthen formulation [PS(π)]. Consider two products i and $i' \in I_1 \cup I_f$ that together require more components than the slot capacity, i.e., $|J(i) \cup J(i')| > C$. If we select both

products i and i', then a certain minimum number of components must be designated as temporary. The following constraint encodes this condition:

$$\sum_{j \in J(i) \cap J(i')} y_j + \sum_{j \in J(i)} z_{ij} + \sum_{j \in J(i')} z_{ij} \geq \min \left\{ |J(i) \cap J(i')|, \frac{|J(i)|+|J(i')|-C}{2} \right\} (x_i + x_{i'}). \quad (4.1)$$

When added to the original problem formulation [P], these constraints eliminate the fractional solution (assigning 1/K th of each product to every machine) described in Section 3. We can include these constraints a priori in formulation [PS(π)] for all product pairs i, i' with $|J(i) \cup J(i')| > B$; however, the formulation size grows by $O(N^2)$ constraints. Instead, with a custom branch-and-cut procedure, we can generate (perhaps, using a heursitic separation routine) and add selected constraints in this class that are violated by the LP solution at each node. We can possibly extend these inequalities for product triples (instead of pairs), and so on.

(iii) The knapsack structure of the workload balancing constraint (3.32) motivates a third class of inequalities. Let TP_I denote the total processing time for all products in the set I_1. Suppose I' is a subset of free products such that

$$\sum_{i \in I'} P_i \quad \leq \quad T - TP_1 \qquad \text{but} \qquad > \quad T - TP_1 - P_{i''} \quad (4.2)$$

for some $i'' \in I_f \backslash I'$. Then, we can impose a valid inequality that limits the number of products selected from the set $I' \cup \{i''\}$ to at most |I'|, i.e.,

$$\sum_{i \in I' \cup i''} x_i \quad \leq \quad |I'|. \quad (4.3)$$

Again, a tailored branch-and-cut procedure can dynamically add these inequalities as needed.

The previous three classes of inequalities exploit the special structure of the constraints in the product selection subproblem. We can also use objective function information to perform problem preprocessing and develop valid inequalities. Consider, for instance, the following method to limit the number of temporary components associated with a product. For a given set of dual values π_i, let $\theta_i = \lfloor \pi_i / b_i \rfloor$ for each product $i \in I_f$ where $\lfloor g \rfloor$ is the largest integer less than or equal to g. Clearly, if an optimal solution in the current branch-and-bound partition selects product i, then no more than θ_i of its components must be temporary; otherwise, we can decrease the net setup cost by setting $x_i = 0$. We can, therefore, add the following valid constraint to the problem formulation at the current node:

$$\sum_{j \in J(i)} z_{ij} \quad \leq \quad \theta_i \qquad \text{for all } i \in I_f. \tag{4.4}$$

Furthermore, suppose we have two products i and i' $\in I_f$ with

$$|J(i) \cup J(i')| - (\theta_i + \theta_{i'}) \quad > \quad C; \tag{4.5}$$

the left-hand side of (4.5) represents the minimum number of permanent slots needed to satisfy (4.4) if we select both products i and i'. Therefore, an optimal solution cannot select both products, i.e., we can add the constraint:

$$x_i + x_{i'} \quad \leq \quad 1. \tag{4.6}$$

Again, we can extend inequality (4.6) to more than two products.

In addition to strengthening the formulation for the product selection problem, other improvement options to consider include heuristically generating a wide variety of "promising" columns before initiating the column generation procedure (our implementation uses only the K columns generated by the initial heuristic solution), and implementing a lower bounding procedure for the PS subproblem so that we generate valid lower bounds on Z_{LP} even if we do not solve the PS subproblem optimally (see Section 3.5). For instance, to generate a lower bound for the PS subproblem, we might consider implementing a Lagrangian relaxation procedure that dualizes the component loading constraints, and applies subgradient algorithm to approximately solve the Lagrangian dual.

Finally, our primary goal has been to generate good product-to-machine assignments, and verify the near-optimality of our feasible solution via lower bounds. To solve the tactical planning problem optimally, we might consider using our algorithm to solve the LP relaxation and generate upper bounds at each node of an overall branch-and-bound procedure. We can use the following branching strategy whenever the procedure cannot fathom a node. Consider two columns with fractional pattern selection values λ_j (in the optimal LP solution at this node) that have at least one product in common. Let product 1 be the common product, and suppose product 2 belongs to the first column but not the second. We then create two successor nodes in the branch-and-bound tree, one requiring that products 1 and 2 must both be assigned to the same machine, and the other requiring them to be assigned to different machines. Using this strategy, at each node we have a set of joint and exclusive*assignment constraints* that reflect the previous branching decisions leading to this node. We can apply the column generation procedure to solve the LP relaxation of the

"constrained" CSC model corresponding to this node and generate feasible solutions. The previously generated columns that satisfy the assignment constraints can serve as the initial set of columns in the restricted master problem, and we add the node's assignment constraints to the PS subproblem; we can easily modify our PS solution procedure to account for these constraints.

5. Concluding Remarks

Electronics assembly poses many important but challenging optimization problems. Much of the management science literature dealing with electronics assembly has focussed on operational problems of single-product cycle time optimization and single-machine setup minimization. The tactical planning issues raised in this paper become increasingly important as electronics companies strive towards agile manufacturing– emphasizing quick response, flexibility, high quality, and cost effective assembly of many different products in small lot sizes–in order to maintain their competitive position and cope with the rapid changes in the product and process technologies. We have presented a model that incorporates both workload balancing and setup optimization issues. Unlike traditional group technology methods that focus on component commonality, our model explicitly considers product demands, production frequencies and machine capacities. The partial setup policy that we considered is practical, and enables us to capture the setup interactions across product families. Managers might find our tactical planning model inherently useful as a framework to address product grouping issues in electronics assembly. The model is quite versatile, capable of incorporating many complicating features of practical problems.

We developed a composite solution procedure that integrates several successful discrete optimization techniques–column generation, lower bounding procedures, and adding valid inequalities–to exploit the special structure of the problem. We showed that even the setup optimization subproblem, which assumes a predetermined assignment of products to a single machine, is NP-hard. We, therefore, focussed on developing a combination of heuristics, lower bounding methods, and enumeration to solve the product selection and setup optimization subproblems. These techniques have direct application to short-term production planning. Although we described the solution method for a simplified problem context (e.g., identical machines, with no prespecified assignments), we can readily modify it for more complex settings such as non-identical machines with multiple placement machines in series in each module. Our limited computational experiments show that the column generation method is effective in generating good upper and lower bounds, but has also identified several opportunities for improvement. Test results for several algorithmic

variants reinforce previous experience regarding effective column generation strategies such as generating more than one negative reduced cost column per iteration. The main computational bottleneck is solving the single machine, product selection subproblem which has a weak linear programmig relaxation. Promising directions to pursue include exploring some of the algorithmic improvement opportunities described in Section 4, and studying the polyhedral structure of the problem.

Acknowledgments:
We thank Professors Thomas Magnanti, Yves Pochet, and Laurence Wolsey for illuminating discussions on solving the tactical planning problem.

Appendix A
Infeasibility Method to Compute a Lower Bound for the Setup Optimization Problem

The method starts with a trial value of 0 for the lower bound Q on the minimum number of temporary slots Q_{min} needed to satisfy the slot capacity constraints, and performs the following steps.

Step 1: **Initialization**

γ = number of permanent slots currently available $\leftarrow C - Q$

I' = current set of candidate products $\leftarrow$ I

J' = current set of remaining components $\leftarrow$ J

Step 2: **Permanent slot reservation test**

Step 2a: If I' or $J \cap J(I')$ is empty, current value of Q is **acceptable**; go to Step 5.
Else, let $i^* \in I'$ be the product with the maximum number of remaining components, i.e., $i^* = \text{argmax}\{|J(i) \cap J'|: i \in I'\}$

Step 2b: If $|J(i^*) \cap J'| \leq Q$, current value of Q is **acceptable**; go to Step 5.

Step 2c: Else, update $\gamma \leftarrow \gamma - (|J(i^*) \cap J'| - Q)$, $I' \leftarrow I' \setminus \{i^*\}$, and $J' \leftarrow J' \cup J(i^*)$;
If $\gamma > 0$, go to Step 2a;
If $\gamma < 0$, then current value of Q is **infeasible**; go to Step 4.
Else ($\gamma = 0$),
if I' or $J \cap J(I')$ is empty, current value of Q is **acceptable**; go to Step 5;
else, go to Step 3.

Step 3: **Permanent component identification test**

Step 3a: Initialize

R = candidate number of permanent slots $\leftarrow C - Q$

J'' = current set of permanent components $\leftarrow \phi$

Step 3b: If I' or $J \cap J(I')$ is empty, current value of Q is **acceptable**; go to Step 5.
Else, let $i^* \in I'$ be the product with the maximum number of remaining components, i.e., $i^* = \text{argmax}\{|J(i) \cap J'|: i \in I'\}$; update $I' \leftarrow I' \setminus \{i^*\}$.

Step 3c: If $|J(i^*) \cap J'| < Q$, current value of Q is **acceptable**; go to Step 5;
Else, if $|J(i^*) \cap J'| = Q$, then every component in $J(i^*) \cap J'$ must be loaded permanently, i.e., update $J'' \leftarrow J'' \cup J(i^*) \cap J'$.

Step 3d: If $|J''| > R$, then current value of Q is **infeasible**; go to Step 4.
Else go to Step 3b.

Step 4: **Infeasibility**

Current trial value of Q is infeasible. Increment Q by 1, and go to Step 1.

Step 5: **Termination**

Current trial value of Q is acceptable. Stop.

Figure 1: Flowchart of Column Generation Algorithm

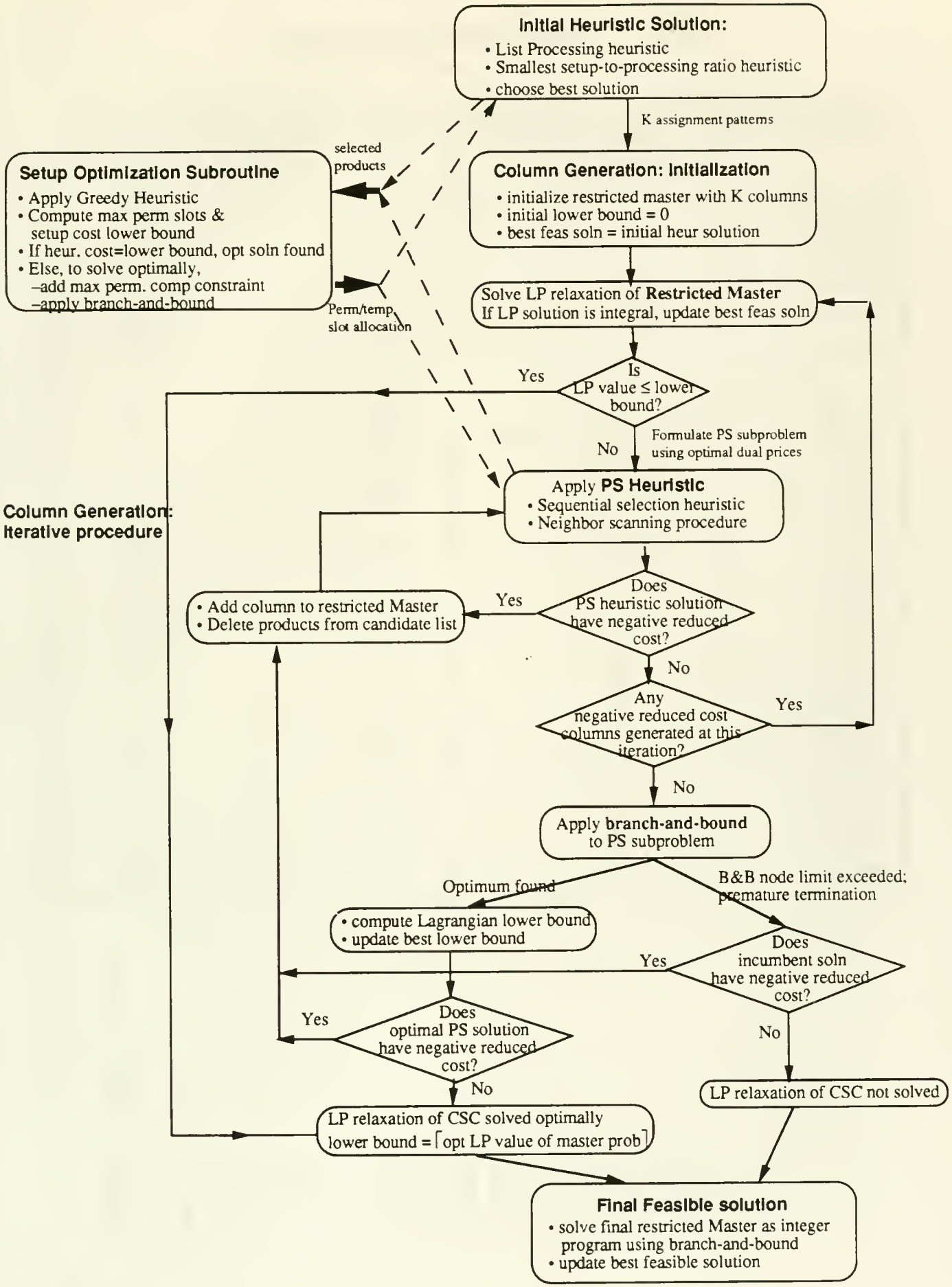


Table 1: Test Problem parameters

Problem Name	No. of Products N	No. of Comp. types M	No. of Machines K	Slot capacity C	Min no. of comp types per board M_{min}	Max no. of comp types per board M_{max}
P10/18A	10	18	3	7	3	5
P10/18B	10	18	3	7	3	5
P09/18A	9	18	3	4	2	3
P09/18B	9	18	3	4	2	3
P09/18C	9	18	3	5	3	3
P15/20	15	20	3	10	5	8
P15/30	15	30	3	15	7	12
P15/40	15	40	3	30	15	25
P15/50	15	50	3	30	15	25
P15/60	15	60	3	30	15	25
P20/20A	20	20	3	6	2	6
P20/20B	20	20	3	7	5	7
P20/20C	20	20	3	8	1	7
P20/20D	20	20	3	9	5	9
P20/20E	20	20	3	10	4	10
P20/30	20	30	3	15	7	12
P20/40A	20	40	3	30	15	25
P20/40B	20	40	4	30	15	25
P20/50	20	50	4	30	15	25
P20/60	20	60	4	30	15	25

Table 2: Computational Results using Standard version of Column Generation algorithm

Problem Name	LCSC Solved?	% Gap†	No. of Columns generated	No. of master iterations	No. of PS subprobs. solved opt using b&b	Total no. of SO subprobs	No. of SO problems using b&b	% of SO probs solved opt by heur§	No. of b&b nodes for final master	% Impr in Initial Heur cost*
P10/18A	yes	0.0	46	5	0	49	22	55	0	100
P10/18B	yes	0.0	52	8	1	74	17	77	0	100
P09/18A	yes	0.0	58	11	1	85	45	47	0	0
P09/18B	yes	0.0	68	12	3	80	51	36	0	12.7
P09/18C	yes	0.0	57	10	1	65	42	35	0	20.6
P15/20	yes	0.0	452	111	66	717	530	26	0	22.2
P15/30	yes	0.0	340	42	16	552	436	21	0	5.4
P15/40	yes	9.70	402	36	8	691	580	16	26	10.8
P15/50	yes	0.0	314	67	46	435	343	21	0	24.4
P15/60	yes	0.0	350	47	29	622	543	12	0	13.1
P20/20A	no	3.68**	688	56	33	852	262	69	73	8
P20/20B	yes	3.9	932	62	44	1004	483	51	14	7.9
P20/20C	yes	5.4	1088	62	13	5761	4066	29	8	28.1
P20/20D	yes	5.2	1019	55	32	1236	913	26	55	0.6
P20/20E	no	2.4**	1429	69	18	2178	2042	6	0	6.3
P20/30	no	4.80**	811	64	32	1084	966	10	6	18.2
P20/40A	no	20.1**	759	72	40	3047	2810	7	4	18.3
P20/40B	yes	3.66	739	66	21	993	850	14	4	15.4
P20/50	no	2.87**	622	49	9	2341	2095	10	0	23.2
P20/60	yes	0.0	483	68	38	591	575	2	0	6.3

† % gap = (Setup cost of best heur solution – Best lower bound)/Best lower bound

* % improvement = (Initial heuristic cost – Final solution cost)/Initial heuristic cost

** Column generation terminated prematurely due to node limit for PS branch-and-bound; % gap measured with respect to best Lagrangian-based lower bound

§ Excluding % of SO problem instances for which SO heuristic finds optimal solution but cannot prove optimality

Table 3: Comparison of different algorithmic options

Problem Name	Version of Algorithm	LCSC Solved?	% Gap†	No of Columns generated	No. of master Iterations	# PS subprob solved opt by b&b	# SO subprob solved by b&b	% SO subprobs olved opt by heur	# b&b nodes for final master
P20/20A	Basic (no PS improvement heuristic)	no	9	93	81	47	19	89	
	Single column generated at each item	yes	9	94	92	48	21	77	59
	Condensed master problem	no	10	110	633	66	31	69	
	Tight proc. limit: $\alpha = 1.05$	yes	7	120	90	52	35	83	
	Loose proc. limit: $\alpha = 1.5$	yes	8	95	84	53	13	92	
	Unlimited depth of search	yes	3.66	1489	57	32	826	47	129
	Max depth of search = 3	yes	3	1443	52	23	718	55	53
	Depth 3 + reduced cost updating	yes	7	1441	53	23	698	54	144
	Standard: Max depth of search=2**	no	3.68	688	56	33	262	69	73
	Reinitialize with best feas. solution	yes	3	632	42	28	261	62	38
P20/20B	Basic (no PS improvement heuristic)	yes	11	118	97	54	37	82	
	Single column generated at each item	yes	12	108	105	57	23	78	8
	Condensed master problem	no	13	505	824	85	229	59	
	Tight proc. limit: $\alpha = 1.05$	yes	17	118	87	38	67	67	
	Loose proc. limit: $\alpha = 1.5$	yes	12	144	126	85	35	86	
	Unlimited depth of search	yes	9	2203	72	44	1238	45	146
	Max depth of search = 3	yes	12	1717	76	48	8682	40	132
	Depth 3 + reduced cost updating	yes	12	1727	65	41	1081	41	112
	Standard: Max depth of search=2**	yes	3.9	932	62	44	483	51	14
	Reinitialize with best feas. solution	yes	3.9	964	73	54	485	52	4
P20/20C	Basic (no PS improvement heuristic)	yes	11	163	135	63	133	55	
	Single column generated at each item	yes	13	146	143	39	105	28	8
	Condensed master problem	no	10	865	1024	69	789	40	
	Tight proc. limit: $\alpha = 1.05$	yes	8	185	147	62	149	54	
	Loose proc. limit: $\alpha = 1.5$	yes	13	162	141	61	121	59	
	Unlimited depth of search	yes	5.4	2122	54	7	3370	84	9
	Max depth of search = 3	no	41	1601	64	9	28216	34	
	Depth 3 + reduced cost updating**	yes	5.4	1611	65	10	1443	24	13
	Standard: Max depth of search = 2	yes	5.4	1088	62	13	4066	29	8
	Reinitialize with best feas. solution	yes	5.4	1658	91	19	33894	22	14
P20/20D	Basic (no PS improvement heuristic)	yes	5	123	97	50	71	67	
	Single column generated at each item	yes	3	117	114	43	58	50	18
	Condensed master problem	no	4	249	780	56	206	38	
	Tight proc. limit: $\alpha = 1.05$	yes	20	117	79	34	90	53	
	Loose proc. limit: $\alpha = 1.5$	yes	5	117	86	32	89	55	
	Max depth of search = 3	yes	2.2	1474	73	30	5011	10	18
	Depth 3 + reduced cost updating**	yes	2.2	1463	66	29	1270	21	18
	Standard: Max depth of search = 2	yes	5.2	1019	55	32	913	26	55
	Reinitialize with best feas. solution	yes	2.2	998	62	32	1262	21	18
P20/20E	Basic (no PS improvement heuristic)	no	4	185	143	22	277	15	
	Single column generated at each item	no	9	196	194	31	185	6	
	Condensed master problem	no	12	925	752	53	1049	18	
	Tight proc. limit: $\alpha = 1.05$	no	6	211	162	23	317	15	
	Max depth of search = 3	no	4	2781	76	14	3377	6	
	Depth 3 + reduced cost updating	no	6	2798	64	7	2941	5	
	Standard: Max depth of search=2**	no	2.4	1429	69	18	2042	6	

† % gap = (Setup cost of best heur solution – Best lower bound)/Best lower bound

** Best method for problem, i.e., lowest %gap, fewest number of PS subproblems solved optimally

References

Ahmadi, J., S. Grotzinger, and D. Johnson. 1988. Component Allocation and Partitioning for a Dual Delivery Placement Machine. *Oper. Res.*, **36**, 176-191.

Ahmadi, R. H., and P. Kouvelis. 1992. An Analytical Framework for the Design of Electronic Assembly Lines: Comparison of Different Design Approaches. ORSA/TIMS Joint National Meeting, San Francisco, November.

Ahmadi, R. H., and H. Matsuo. 1992. A Mini-Line Approach for Pull Production. Working Paper, Anderson Graduate School of Management, UCLA.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey.

Baker, K. R. 1974. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, New York.

Ball , M. O., and M. J. Magazine. 1988. Sequencing of Insertions in Printed Circuit Board Assemblies. *Oper. Res.*, **36**, 192-201.

Bard, J. F., R. W. Clayton, and T. A. Feo. 1989. Optimizing Machine Setup and Component Insertion in Printed Circuit Board Assembly. Working Paper, University of Texas, Austin.

Barnhart, C., E. Johnson, R. Anbil, and L. Hatay. 1992. Solution Techniques for Long-Haul Crew Assignment Problem. Working Paper, Cuomputational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia.

Bradley, S. P., A. C. Hax, and T. L. Magnanti. 1977. *Applied Mathematical Programming*, Addison -Wesley, Reading, Massachusetts.

Carmon, T. F., O. Z. Maimon, and E. M. Dar-El. 1989. Group Set-Up for Printed Circuit Board Assembly. *Intl. J. Prod. Res.*, **27**, 1795-1810.

CPLEX Optimization, Inc. 1991. *Using the CPLEX Linear Optimizer*, version 1.2. CPLEX Optimization, Inc., Incline Village, Nevada.

Daskin, M. S., O. Maimon, and A. Shtub. 1991. A Branch and Bound Algorithm for Grouping Components in Printed Circuit Board Production. Working Paper, Department of Civil Engineering, Northwestern University, Evanston, Illinois.

Desrochers, M., J. Desrosiers, and M. Solomon. 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, **40**, 342-354.

DeWitte, J. 1980. The Use of Similarity Coefficients in Production Flow Analysis. *Int. J. Prod. Res.*, **18**, 505-514.

Drezner, Z., and S. Nof. 1984. On Optimizing Bin Packing and Insertion Plans for Assembly Robots. *IIE Trans.*, **16**, 262-270.

Francis, R. L., H. W. Hamacher, C-Y. Lee, and S. Yeralan. 1989. On Automating Robotic Assembly Workplace Planning. Research Report, Industrial and Systems Engineering Department, University of Florida, Gainesville.

Frenk, J. B. G., and A. H. G. Rinnooy Kan. 1984. The Asymptotic Optimality of the LP Rule. Report 8418/O, Econometric Institute, Erasmus University, Rotterdam.

Gavish, B., and A. Seidmann. 1987. Printed Circuit Boards Assembly Automation–Formulations and Algorithms. *Proc. ICPR*, Cincinatti, Ohio.

Graham, R. L. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Math.*, **17**, 263-269.

King, J. R. 1980. Machine-component Grouping in Production Flow Analysis: An Approach using a Rank Order Clustering Algorithm. *Int. J. Prod. Res.*, **18**, 213-232.

King, J. R., and V. Nakornchai. 1982. Machine-component Group Formation in Group Technology: Review and Extension. *Int. J. Prod. Res.*, **20**, 117-133.

Lasdon, L. S. *Optimization Theory for Large Systems*. Macmillan, New York

Lawler, E. L., J. K. Lenstra, and A. H. G. Rinnooy Kan. 1982. Recent Developments in Deterministic Sequencing and Scheduling: A Survey. In *Deterministic and Stochastic Scheduling*, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan (eds.), Reidel, Dodrecht.

Lofgren, C. B., and L. F. McGinnis. 1986. Optimizing Electronics Assembly System. *Proc. IIE Elec. Assembly Conf.*.

Magnanti, T. L., J. F. Shapiro, and M. H. Wagner. 1976. Generalized Linear Programming Solves the Dual. *Management Science*, **22**, 1195-1203.

Magnanti, T. L., and R. T. Wong. 1981. Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria. *Operations Research*, **27**, 464-482.

Nemhauser, G. L., and L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York.

Pochet, Y., and L. A. Wolsey. 1992. Personal communication.

Prasad, R. P. 1989. *Surface Mount Technology: Principles and Practice*. Van Nostrand Reinhold, New York

Rajagopalan, R., and J. L. Batra. 1982. Design of Cellular Production Systems: A Graph-theoretic Approach. *Int. Journal of Prod. Res.*, **13**, 567-579.

Skinner, W. 1974. The Focused Factory. *Harvard Business Review*, 113-122.

Vance, P. H., C. Barnhart, E. L. Johnson, and G. L. Nemhauser. 1992. Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound. Working paper, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia.

Vanderbeck, F. A Decomposition Approach for Parallel Machine Assignment and Setup Minimization in Electronics Assembly. Master's thesis, Operations Research Center, Massachusetts Institute of Technology, Cambridge.

Basement

Date Due