Ann Oper Res (2013) 203:295–323
DOI 10.1007/s10479-012-1167-5

A GRASP for simultaneously assigning and sequencing product families on flexible assembly lines

Susan K. Heath · Jonathan F. Bard · Douglas J. Morrice

Published online: 13 June 2012

Abstract This paper introduces a new model and solution methodology for a real-world production scheduling problem arising in the electronics industry. The production environment is a high volume, just-in-time, make-to-order facility with volatile demand over many product families that are assembled on flexible lines. A distinguishing characteristic of the problem is the presence of non-traditional sequence-dependant setup costs, which complicate our ability to find high-quality solutions. The scheduling problem arose when product variety exceeded the mix that the existing lines could accommodate. A nonlinear integer programming formulation is presented for the problem of minimizing setup costs, and a greedy randomized adaptive search procedure (GRASP) is developed to find solutions. To select the GRASP parameter values, an efficient, space-filling experimental design method is used based on nearly orthogonal Latin hypercubes. The proposed methodology is tested on actual factory data and compared to a prior heuristic presented in the literature; our heuristic provides a cost savings in 7 out of the 10 cases examined, and an average improvement of 17.39 % which is shown to be highly statistically significant. This improvement is due in part to the introduction of a pre-processing step to determine preferential and non-preferential line assignment information.

Keywords GRASP · Production sequencing · Nearly orthogonal Latin hypercubes · Nonlinear integer programming

S.K. Heath (✉)
Naval Postgraduate School, 555 Dyer Road, IN-233, Monterey, CA 93943, USA
e-mail: skheath@nps.edu

J.F. Bard
Graduate Program in Operations Research and Industrial Engineering, The University of Texas, Austin, TX 78712, USA

D.J. Morrice
McCombs School of Business, The University of Texas, 1 University Station B6500, Austin, TX 78712, USA

1 Introduction

The sequencing problem addressed in this paper derives from a production scheduling application at a mass-customization, assemble-to-order electronics manufacturing facility. Although this real-world scheduling problem can be roughly described as consisting of the same three general steps as all scheduling problems, that of (1) the assignment of jobs to production lines, (2) the sequencing of the jobs on the lines, and (3) the scheduling of the exact times each job will be placed on and taken off each line within this sequence (Pinto and Grossmann 1998), it has significant differences from scheduling problems considered in the existing literature. To begin with, jobs are grouped by families and, rather than scheduling individual jobs on a production line, we must schedule each family to be on the line, meaning that all parts required to build any job in that family are placed on the production line. Our problem also has multiple flexible production lines, where each line can hold the parts for multiple families simultaneously so that jobs from any of these families may be produced without requiring a setup. Setups are only required when jobs from a different family, whose parts are not on the line, need to be produced.

Allahverdi et al. (2008) provide a good survey of production scheduling research that pertains to minimizing setup times or costs; they also introduce new notation to describe these problems. Using their notation, the problem modeled here would appear to be denoted as $P/SC_{sd,b}/TSC$ which refers to a problem of scheduling batches of jobs with sequence-dependant setup costs ($SC_{sd,b}$) on parallel identical machines (P) with the goal of minimizing total setup costs (TSC). Although this is the best available notation for our problem, using it would mask two additional complications due to the factory's use of product families and flexible lines: (1) A setup on these flexible lines consists of removing one family from the line while the line continues to produce jobs from the families remaining on the line during the setup. This means that the sequence-dependant setups not only depend on the family coming off the line and the family going on the line (traditional sequence-dependant setups) but also on the families remaining on the line during the setup. It also results in no downtime for setups. (2) Since multiple families are placed on a line simultaneously, the length of time a family needs to spend on a line is not determined exclusively by the 'batch size,' i.e., the number of jobs in that family, that needs to be produced. The implications of these, and other, important differences are detailed in subsequent sections, but these differences make modeling and solving the problem much more complex. To get a sense of how much more difficult the problem becomes with these complexities, we note that the number of family nodes required in the graph representation of our test problem is 100 times greater than the number of family nodes that would be required if each production line held only one family and the sequence-dependant setup costs were traditional in nature. We are not aware of any production scheduling research that tackles problems with these characteristics without resorting to serial decomposition.

In fact, the real-world problem investigated in this paper was previously addressed by Monkman et al. (2005, 2008) who decomposed it into three separate scheduling steps in which each production line was sequenced independently. Their decomposition approach in the 2008 paper, herein referred to as the MMB heuristic, provides us with a benchmark against which to compare our solutions. In this paper, we take an integrative approach that assigns product families to lines and sequences them simultaneously. Our focus is on the combined assignment and sequencing steps, which determines the total setup costs. The third step of the scheduling procedure, which involves specifying when setups occur on each line, does not affect setup costs and is therefore outside the scope of the proposed scheduling methodology. However, since the output of the first two scheduling steps feeds

into the third scheduling step, some considerations must be made in the assignment and sequencing steps in anticipation of the time scheduling step. We present these considerations where they appear in the model.

Given that sequence-dependent machine scheduling problems are strongly NP-hard and our problem is a generalization of such problems, it is also strongly NP-hard. If our current model could be solved to optimality it would dominate the MMB heuristic. Although there has been much progress in solving large instances of difficult combinatorial optimization problems, our initial experience with commercial software indicated that our new nonlinear integer programming (NLIP) model is not amendable to exact methods. Therefore, we developed a greedy randomized adaptive search procedure (GRASP) (Feo et al. 1991; Feo and Resende 1995) to find solutions to the assignment and sequencing steps of the scheduling problem. In order to achieve the best possible performance of our GRASP, we designed and executed an experiment using a nearly orthogonal Latin hypercube (NOLH) design (Cioppa 2002; Cioppa and Lucas 2007) to determine the best parameters for the GRASP. When compared to the MMB heuristic, the new GRASP provided a reduction in setup costs in 7 out of 10 cases, with an average reduction of 17.39 % across all 10 cases. Using a paired t-test, the average performance improvement of the GRASP results over the MMB results was found to be highly statistically significant with a p-value of 0.0075.

The primary contributions of this paper are threefold: (1) We present a real-world production sequencing problem that involves complexities leading to a new type of optimization model. Our model most closely resembles an extension of the traveling purchaser problem (Laporte et al. 2003), which can be described as a capacitated multiple traveling purchaser problem, an example of which, to our knowledge, has not yet appeared in the literature; (2) We describe a new GRASP that can accommodate nonlinear constraints. The authors are only aware of a few previous instances where a GRASP was used for a nonlinear problem (Bard 1997; Mavridou et al. 1998; Shen et al. 2008), but our problem is the only known case where the nonlinear constraints cause feasibility issues during the solution construction phase of the GRASP, a critical difficulty that must be overcome for our problem; and (3) We demonstrate for the first time the use of an NOLH experimental design to determine the best parameters for a GRASP.

In the next section, we outline the production scheduling problem that motivated our research. Section 3 contains the development of the NLIP formulation. In Sect. 4, we describe the GRASP and provide an example to demonstrate how the procedure works. The NOLH experiments conducted to arrive at the parameter values used in the subsequent analyses are discussed in Sect. 5 along with the results obtained by comparing its performance with the MMB heuristic. Several observations are made in Sect. 6 on the effectiveness of the two procedures and recommendations are given for future research.

2 Production scheduling problem description

In this section we provide all the details of the production environment, the product, and the production line configuration that make this a particularly complex production scheduling problem. We then give a concise problem statement that provides the foundation for the modeling and solution sections that follow. Additional rules and policies required by the manufacturer regarding their production schedules are presented and incorporated after the initial model is formulated.

2.1 Production environment, products, and production line configuration

The sponsoring company manufactures electronics products in a make-to-order fashion. In doing so, they use just-in-time principles for bringing in raw materials, and promise their customers that orders will be delivered within a short period of time. Demand is highly variable and daily demand is comprised of a large volume of jobs, numbered in the tens of thousands. To accommodate the high variability in demand and prevent the production lines from running out of jobs to work on, the facility allows a small backlog of jobs to be carried over from one eight-hour shift to the next. This backlog consists of jobs that have been received by the factory but not yet produced at the end of a shift. The size of the backlog is allowed to vary in order to absorb the fluctuations in demand, but the target average backlog level is approximately one shift's worth of production.

Orders arrive daily to the sales organization from a multitude of customers for many different products. When the orders arrive they are given a timestamp and then processed by the sales group. If indicated, an order is also processed by the engineering group. Once the orders are fully processed, they are transferred to the production facility where they are separated into jobs, with one job being one product. Jobs are downloaded into the factory's production system at the beginning of a shift and again every two hours during the shift.

Each product consists of one chassis (a case) and the component parts that are to be assembled within the chassis. Customers are able to customize their orders by selecting the chassis as well as the mix of parts to go in it. This results in a very high variety of possible products that may be ordered. Therefore, the products are grouped into product families based on the chassis. Each product family's parts include the family's unique chassis type as well as all of the component parts from which a customer is allowed to choose when ordering a product with that chassis. No two product families use identical groups of component parts; some component parts are used by many families, some are used by only one family, and many in between. Although there is a high variety of possible products, each production line is configured in such a way that the products can flow down the line at a fairly constant pace. Therefore, all jobs can be assumed to take roughly the same amount of time to assemble.

In order to accommodate the wide variety of products, the facility was designed with multiple, identical, flexible production lines running in parallel. Each flexible line can hold all the parts for several families simultaneously, so a line can be configured to produce products from any of several families without requiring a setup. More specifically, each production line has designated floor space at its head for several pallets of chassis to be staged there. This allows the line worker to have easy access to whichever chassis is needed for the next job to be produced. Similarly, boxes of the component parts for these families are placed alongside the line so the needed parts for the job being produced can be readily accessed. A depiction of a production line with three chassis areas, each of which would hold a pallet of chassis for one family, is shown in Fig. 1. Just as there are spaces for only a few chassis types to be placed on one line, there is limited space for component parts as well, so only the parts for the families whose chassis are on the line are placed alongside the line.

One special feature of this production line configuration is that it allows one family's chassis to be removed from the line, as well as all the component parts used only by that family, without interfering with the production of jobs from the other families on the line. This means that a setup can take place without causing production downtime. Specifically, a setup consists of removing one family's parts from the line that are not also required by any of the families remaining on the line, and adding another family's parts to the line that are not already on the line. It is important to note, however, that downtime will only be avoided

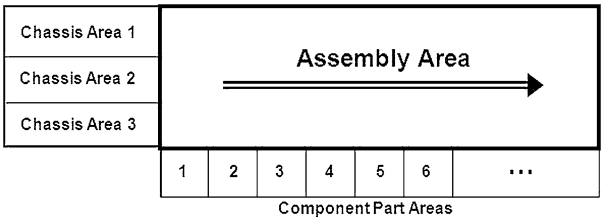


Fig. 1 Example of a single production line configuration with three chassis areas

if there is enough demand for the families that remain on the line during the setup to keep the line busy during the setup. In addition, since each chassis area can be accessed by the line worker during production, there is no advantage to having one family's chassis in more than one chassis area; in fact, it would be detrimental since the production line would then accommodate fewer families.

The number of different families produced by the facility was historically small enough that all families could be placed on at least one line simultaneously so any product from any family could be produced without requiring a setup. Each incoming job could be routed to the line (or one of the lines) that held that job's family, and jobs routed to the same line were produced in (FCFS) order based on the job's timestamp. With this configuration, setups were only needed in the relatively rare instances of drastic shifts in demand, new product family introductions, or the end-of-life of a product family.

2.2 Problem statement

Although the factory was originally designed to operate without regularly scheduled setups, in a very short period of time the number of product families increased sharply, nearly doubling in the span of months. The first consequence was that setups were required since there was no longer enough space to place all families on lines simultaneously. In addition, it became clear that it was impossible to continue producing jobs in strict FCFS order since there was simply not enough time in a shift to perform all the setups this would require. The factory responded by effectively having two queues per line: a production queue for jobs whose family was currently on the line, and a hold queue for jobs whose family was assigned to the line but not currently on the line. Within each queue, jobs could be positioned in FCFS order; however, with the addition of a hold queue, some jobs in the production queue would be produced before jobs in the hold queue having an earlier timestamp.

Setups were originally introduced on an ad-hoc basis; when the production queue became relatively small, or orders in the hold queue grew too old, a setup was performed. However, without a formal plan for when to schedule these setups four problems began occurring: (1) lines sometimes experienced unplanned downtime during a setup because they ran out of jobs for the families remaining on the line during the setup; (2) the amount of work required for setups was onerous when too many setups were called for; (3) some jobs remained backlogged in a hold queue for too long resulting in an unacceptably late delivery to the customer; and (4) lines occasionally ran out of parts for jobs, and there were too many of other parts clogging the facility, because the JIT parts ordering system could not supply the correct parts when setups were not predictable. This motivated the need for a production scheduling method that could be used to determine when families should be placed on lines while considering the many special requirements of the facility.

To better explain the type of schedule that the facility desired, Fig. 2 gives a pictorial representation of a sample schedule for a single production line. The assumptions for this example are that the line has three chassis areas, there are five families assigned to the line,

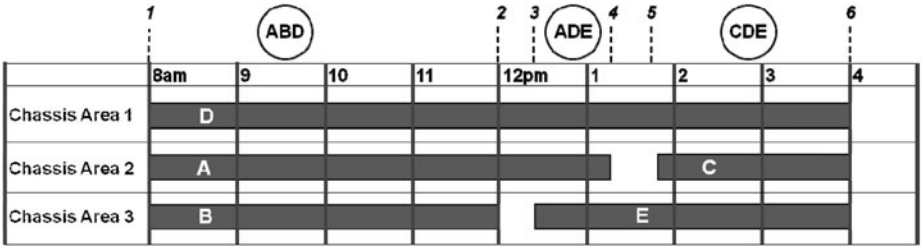


Fig. 2 Sample schedule for product families A–E on a single line that can accommodate three families

and these families, listed from highest demand to lowest demand, are D, A, B, E, and C. In the figure there are three rows, one representing each chassis area on the line. (Recall, however, that there is only one assembly area on the line, as shown in Fig. 1, so as long as there is at least one family on the line and there is demand for that family, the line can keep running.) The x-axis is time, and each bar with a family name in it shows the time during which that family's chassis is in the designated chassis area and the component parts are also on the line. A gap between the bars in a row indicates that a setup will occur at that time, with the family to the left of the gap being removed from the line and the family to the right of the gap being added. We can also see, by looking at the bars above and/or below a gap, which families will remain on the line during that setup. For example, during the first setup, which starts at 12:00 pm, family B is removed and family E is added, and families A and D remain on the line so the line can continue processing jobs from these two families during the setup. The circles at the top represent the groups of three families that are on the line between setups and provide a visual example of the nodes we define in Sect. 3.1.

The primary goal of the facility in constructing the production schedule was to minimize the amount of work required for all setups during the shift. This may seem counterintuitive since a common goal in scheduling is to minimize setup times. However, since setups do not cause production downtime, the only real goal or constraint regarding setup times is that there must be enough time in the shift to perform all the required setups on each production line (so the sum of the setup times on a single line cannot exceed the length of the production shift). The amount of labor required for setups was not constrained. However, the factory's desire was to minimize the total required setup work since any excessive work not only requires more labor but also causes extreme congestion in the areas between and in front of the production lines where the setups actually take place. The best measure of work content in a setup was the total number of parts that needed to be changed during the setup, which we use as the setup cost. The fact that work content and setup times are positively correlated does mean that achieving the objective of minimal setup costs will also achieve lower total time spent on setups. However, one additional constraint is added to the model, described in Sect. 3.2, to ensure that all required setups can be performed during the shift.

3 Model formulation

From the description of the scheduling problem, it should be clear that product families are the items to be modeled rather than individual jobs, since setups occur only when the families on a line change. Nevertheless, some consideration for the volume of individual jobs to be produced per family must be given. One additional complication makes this more difficult: demand is not fully known when the schedule must be constructed. At that time, the

only demand known for sure for each family is the number of jobs in the backlog left over from the previous shift (backlogged demand) which can vary significantly from shift to shift. Because orders arrive every 2 hours during the day as well, some demand will arrive to the factory and be produced on the same day. To account for both the backlog and the unknown amount of demand that will arrive during the production shift, we used the actual backlog plus the expected demand over one shift (based on the historic mean) as the total demand for each family that should be considered during the scheduling process. In the context of the model the term *demand*, when used alone, will denote this sum of the backlogged demand and the expected demand.

For our problem, the three production scheduling steps are described as follows: Step 1 consists of assigning families to production lines; step 2 consists of sequencing sets of families on each production line; and step 3 consists of specifying when the setups should occur. Although all three steps are required to develop a complete production schedule, only steps 1 and 2 affect the setup costs. Therefore, our approach is to model steps 1 and 2 using a single optimization model with the objective of minimizing setup costs. On the other hand, it is important to note that the output of each step affects the next: the assignments made in step 1 will constrain the possible sequences in step 2, and the sequences in step 2 will constrain the times at which the setups can occur in step 3 and therefore constrain how much time a family could be allowed to remain on the line. The relationship between steps 1 and 2 is what motivates our combined model, and the subsequent relationship to step 3 requires some consideration of time-scheduling in our assignment and sequencing model. These considerations will be discussed further where they appear in the model.

3.1 Model components

In this section, we develop an optimization model for the assignment and sequencing portions of the scheduling problem, incorporating all the information given above. In the next section we discuss several additional rules imposed by the manufacturer and the constraints that must be added to the model to enforce them.

Taking a network approach, we represent the problem as a set of paths in a graph $G = (V, A)$, where the nodes in V represent each possible set of families that can be on a single line simultaneously. In a solution, each path through G represents the sequence of sets of families to be placed on one line during a shift. Therefore, the final number of paths equals the number of production lines being scheduled. By design, all families that appear in at least one node on a line's path are the families that are assigned to that line.

The cost for each arc in set A is calculated as the cost incurred when a change is made on a line from the families in the node at one end of the arc to the families in the node at the other end of the arc. Since the cost is the number of parts that the sets of families across the two nodes do not have in common, the graph is undirected. However, each path through the graph will be directed since each path's initial node in V is specified by the set of families remaining on the line from the prior shift. Due to the nature of the setup costs, the arc costs in G satisfy the triangle inequality (proof is in the Appendix).

These costs are non-traditional due to the fact that each setup depends on factors beyond the characteristics of the families being removed from and added to the line. For example, say a line can accommodate two families, where family A is the family being removed from the line, family B is being added, and the number of parts that A and B do not have in common is 25. The arc's setup cost depends not only on A and B, but also on the family remaining on the line (the second family currently on the line). If family C remains on the line during the setup and has no parts in common with families A and B; then the setup cost

is 25. However, if family D uses all the same parts as families A and B and is the family on the line during the setup, the setup cost is 0 because family B's parts are already on the line and family A's parts must remain on the line for family D. When we consider the fact that the family remaining on the line must have been added to the line earlier on the path, we see that each arc's setup cost has dependencies that go beyond its origin node. Therefore, each node that adds a new family can influence later arc costs.

Before introducing the model, it is important to note that at the beginning of a shift, a line will still have families remaining on it from the end of the previous shift. Therefore, each line's path will begin at a different node in V. However, there is no restriction on which node any line's path ends; and a line's path would rarely end where it began. Since we do not know where any line's path will end, it would be difficult to write the balance of flow constraints correctly without an additional construct. We therefore introduce a dummy node called the "depot" where each line's path needs to begin and end. The arc costs to and from the depot will all be zero, but additional constraints are needed to ensure that each line's path goes from the depot to the node containing the families that are on the line at the beginning of the shift (remaining there from the end of the previous shift). Defining K as the set of all production lines, the node that contains the families remaining on line k from the previous shift will be denoted by v_k for all $k \in K$. The depot node will be denoted by O, the set of nodes $V \cup O$ will be denoted by V^+, and the graph with the additional node and additional arcs will be denoted by G^+. Additional notation will be defined as it is used.

With the goal of minimizing the total setup costs, our objective function is

$$\text{Minimize} \sum_{k \in K} \sum_{j \in V^+} \sum_{i \in V^+} x_{ijk} c_{ij} \tag{1a}$$

where x_{ijk} is a decision variable equal to 1 if the arc from node i to node j is included on the path for production line k, and 0 otherwise; $x_{iik} = 0 \ \forall i \in V^+$; and c_{ij} is the cost of the arc from node i to node j.

The first set of constraints formalizes the idea of the assignment of families to lines. Specifically, we need to ensure that a line's path only includes nodes where all families in the node are assigned to that line. To do this, we constrain the arcs on a path to include only arcs that go to nodes containing families assigned to the line. We define the decision variable y_{pk} to be equal to 1 if product family p is assigned to line k, and 0 otherwise. This set of constraints can then be written as

$$\sum_{i \in V^+} \sum_{j \in S_p} x_{ijk} \leq \left|V^+\right| y_{pk}, \quad \forall p \in P, \ k \in K \tag{1b}$$

where P is the set of all product families and S_p is the set of all nodes in V that includes product family p.

The flow balance requirements make up the next few sets of constraints.

$$\sum_{i \in V^+} x_{Oik} = 1, \quad \forall k \in K \tag{1c}$$

$$x_{Ov_k k} = 1, \quad \forall k \in K \tag{1d}$$

$$\sum_{i \in V^+} x_{ijk} = \sum_{i \in V^+} x_{jik}, \quad \forall j \in V, \ k \in K \tag{1e}$$

Constraints (1c) ensure that exactly one path leaves the depot for each line in K, and constraints (1d) ensure that the path for line k goes from the depot to the node with the families remaining on the line from the prior shift, v_k. Flow balance is maintained through all nodes on each line's path by constraints (1e).

The next set of constraints eliminates subtours.

$$\sum_{i\in S}\sum_{j\in S} x_{ijk} \le |S| - 1, \quad \forall S \subseteq V,\ 2 \le |S| \le |V|,\ k \in K \tag{1f}$$

Note that a production line's full path, which includes its depot node, is, by definition, a subtour, so the subtour elimination constraints (1f) only apply to the nodes in V. In addition, they have to be written for all $|S| \le |V|$ rather than for all $|S| \le |V|/2$, which is more common, because the reduced set of constraints is not guaranteed to prevent subtours when all the nodes in V are not required to be visited in a solution.

Next, we consider the demand for each family assigned to each line. The demand for family p will be denoted by d_p, with the unit of measure being jobs, and is equal to the number of jobs that must be assigned to a production line. Although backlogged demand for a family may be zero at times, the expected demand will always be positive, so d_p, being the sum of these values, is always positive. We also note that any jobs remaining in the backlog do not retain their line assignment from the prior shift; backlogged jobs get a new line assignment when a new schedule is constructed.

Letting the decision variables z_{pk} equal the number of jobs of family p assigned to line k, the first three sets of demand-related constraints are

$$\sum_{k\in K} z_{pk} = d_p, \quad \forall p \in P \tag{1g}$$

$$d_p y_{pk} \ge z_{pk}, \quad \forall p \in P,\ k \in K \tag{1h}$$

$$\sum_{p\in P} z_{pk} \ge L^{cap}, \quad \forall k \in K \tag{1i}$$

where (1g) ensure that all demand for each family gets assigned to a line, (1h) ensure that no demand is assigned to a line whose family has not been assigned to that line, and (1i) ensure that the number of jobs assigned to a line is at least as large as the line's capacity, denoted by L^{cap}, in order to prevent unplanned downtime due to too little demand on the line. Recall that the factory always maintains a backlog so constraints (1i) will not cause infeasibility. We also note that since d_p is always positive, constraints (1g) and (1h) guarantee that every family will be assigned to a line. Since the portion of d_p that makes it positive is an expected value, it is actually possible to have zero backlogged demand and zero realized demand over a shift for a given family. In this situation the setups performed to place this family on the line during that shift would not have been necessary. This is a consequence of the demand uncertainty at scheduling time. The factory was aware that maintaining the requirement that a family be assigned to a line even when backlogged demand was zero could potentially lead to an unnecessary setup (in the event that the realized demand was also zero), but they preferred this risk to the alternative of not assigning that family to a line and forcing any arriving demand to be backlogged an extra day.

The final set of demand-related constraints results from consideration of how the production line sequences provided by the model's solution constrain the time-scheduling step (step 3 of schedule construction). To understand these constraints it is necessary to understand in part the time-scheduling model that underlies step 3. To explain this further, we refer back to Fig. 2. Notice the inclusion of italicized numbers above the time line markers. The numbers *1* and *6* indicate the start and end of the shift, respectively. The *2* indicates the time that the first setup begins and the *4* indicates the time the second setup begins, with the *3* and *5* indicating the end of these respective setups. Once the sequence is determined, the time each individual family can spend on the line can no longer be adjusted freely within the shift.

To demonstrate this, consider family B. Since family B appears in only the first of the three nodes in the sequence, the only way to maximize the amount of time it spends on the line (increase the duration between *1* and *2* in the figure) is to compress the durations from *3* to *4*, and from *5* to *6*, to zero. (The durations from *2* to *3* and from *4* to *5* cannot be compressed since they constitute setups and eliminating them would change the sequence.) So, at the extreme, the maximum time B could be allowed to spend on the line is the total length of the shift (here eight hours) minus the time required for the two setups. Note that this would result in eliminating any opportunity to produce jobs in families C and E. If family B comprised a vast majority of the demand assigned to the line, this may be necessary to avoid excessive backlogging of family B jobs; however, that would cause all demand associated with families C and E to be backlogged. In addition, it might cause the line to produce so much of the demand for families A and D earlier in the shift that there would not be enough left to keep the line running during the two setup periods at the end of the shift, resulting in unplanned downtime costs. Looking at family D there is no analogous problem, though, because it appears in all the nodes on the line's path so it can spend the entire shift on the line.

This suggests that some consideration needs to be given to the relative amounts of demand for each family assigned to a line when determining how many nodes a family appears in on that line's path. To ensure that each family has enough opportunity for its jobs to be produced, during the time-scheduling step we allot the time that a family spends on the line in proportion to the demand associated with that family. To allow the flexibility for the time-scheduling step to be able to do this, it is sufficient to ensure that the proportion of nodes in a line's path in which a family appears is at least as large as the proportion of the total line's demand for that family. For example, consider a production line that can accommodate three families simultaneously and has five families assigned to it with the following demands: $d_{\mathrm{A}} = 5$, $d_{\mathrm{B}} = 10$, $d_{\mathrm{C}} = 15$, $d_{\mathrm{D}} = 25$, $d_{\mathrm{E}} = 45$ (total demand $= 100$). If the path for this line visits three nodes, then family E will need to appear in at least 45 % of those nodes, i.e., in at least two of those nodes. For the other families, it will be sufficient for them to only appear, at a minimum, in just one of the nodes, since each of their demand percentages on the line is less than $1/3$ or 33.3 %. The following constraints ensure this condition holds.

$$\frac{\sum_{i\in S_p}\sum_{j\in V^+} x_{ijk}}{\sum_{i\in V}\sum_{j\in V^+} x_{ijk}} \geq \frac{z_{pk}}{\sum_{l\in P} z_{lk}}, \quad \forall p \in P,\ k \in K \tag{1j}$$

Since the number of nodes visited on a line's path, containing each family and in total, are all functions of the decision variables, the ratio on the left-hand side of (1j) is nonlinear. In addition, the amount of demand assigned to a line for each family and in total are all functions of the decision variables, so the ratio on the right-hand side is nonlinear as well. Constraints (1j) also ensure that a line's path includes at least one node containing each of the families assigned to that line.

To complete the basic formulation, constraints (1k) and (1l) are included to ensure that all the x and y decision variables are binary, and all the z decision variables are integer.

$$x_{ijk} \in \{0, 1\}, \quad \forall i, j \in V^+,\ k \in K \tag{1k}$$

$$y_{pk} \in \{0, 1\}, \quad z_{pk} \in \{0, 1, 2, \ldots, d_p\}, \quad \forall p \in P,\ k \in K \tag{1l}$$

Model (1a)–(1l) satisfies the specific details and complications set forth in the problem description. As mentioned, however, the company had additional rules and policies that required several more constraints. The details are now given.

3.2 Additional rules and associated constraints

There are three additional rules at the facility that must be reflected in the model. The first is that only one family can be removed from a line during each setup, and consequently only one can be added. This rule is motivated by the goal of keeping the production line running at all times, even during setups. A line can continue production during setups as long as there are enough jobs remaining for the families that stay on the line during the setup. Because demand is not fully known when the schedule is constructed, whether or not a line will run out of demand during a setup cannot be predicted. The fewer the number of families that remain on the line during a setup, the more likely that the line will run out of jobs to produce during a setup and incur unexpected downtime costs. The company placed a much higher priority on avoiding unexpected downtime than any additional setup efforts this rule might occasion. Its effect on the model is that the only arcs allowed in set A are arcs that connect two nodes that differ by only one family.

The second additional rule places bounds on the number of families that can be assigned to a line. A natural lower bound is the number of chassis areas on the line, since allowing a chassis area to be left empty or allowing a family to be placed in two chassis areas will most likely cause an additional setup somewhere else. This lower bound is denoted by L^{low}, and the number of chassis areas on a line is the value that the manufacturer chose. The upper bound is based on the amount of time setups take in relation to the length of a shift. The aim is to ensure that (1) there will be sufficient time for all setups to occur, and (2) that the set of families in each node will be given enough time on the line to make the setup worthwhile. This upper bound is denoted by L^{up}. The value the company chose for L^{up} was roughly 2.5 times L^{low}, which they felt would keep the number of setups on any line to a manageable number and therefore keep the time required for all setups on a line well under the length of a shift. To enforce these bounds, we introduce the following constraints.

$$\sum_{p\in P} y_{pk} \le L^{up}, \quad \forall k \in K \tag{1m}$$

$$\sum_{p\in P} y_{pk} \ge L^{low}, \quad \forall k \in K \tag{1n}$$

The third additional rule is designed to maintain the ability to reassign jobs from one production line to another during a shift without changing the schedule. This gives the shift supervisor the ability to make last minute changes if realized demand is drastically different than expected for one or more families. The rule imposed by the company was that all families with high expected demand must be placed on at least two lines and each line must have at least two high-demand families assigned to it. With regard to the distribution of demand across families, the factory observed that the 80/20 rule applied with roughly 20 % of the families comprising about 80 % of the demand. Accordingly, the top few families were labeled *high runners* and were required to be assigned to at least two lines. This rule had the additional benefit of ensuring that families whose demand well exceeded the capacity of a single production line would be assigned to more than one line. Defining Q as the subset of families that were deemed high runners, the corresponding constraints are as follows.

$$\sum_{p\in Q} y_{pk} \ge 2, \quad \forall k \in K \tag{1o}$$

$$\sum_{k\in K} y_{pk} \ge 2, \quad \forall p \in Q \tag{1p}$$

This completes the model.

3.3 Prior approaches and related literature

Loveland et al. (2007) were the first to address the production scheduling problem described above. As a first cut, they assumed that setup costs were roughly sequence independent. This led to the development of an IP formulation of the assignment step and the use of heuristics and linear programming to solve the sequencing and time-scheduling steps, respectively. A highlight of the paper was the discussion of how the approach was implemented by the company.

The second approach, the MMB heuristic, appeared in Monkman et al. (2005, 2008). There, the authors explicitly modeled the true sequence-dependant setup costs but decomposed the problem into the three separate scheduling steps using a different optimization model for each. Since the setup costs depend on the first two steps of the scheduling problem and the second step depends on the first, the setup costs could not be minimized. In their methodology, each production line was sequenced independently. They then demonstrated how this approach yielded lower setup costs than the method presented by Loveland et al. (2007) for all cases studied.

In theory, our approach should dominate the other two since first assigning families to production lines and then separately sequencing them on each line does not even guarantee local optimality. However, (1a)–(1p) results in a much larger model than can be solved to optimality so any heuristic solution approach needs to be compared with the performance and results provided by the MMB heuristic.

With respect to the general field of production planning and scheduling, it has long been recognized that some amount of decomposition is the only practical way to find solutions to most real-world problems. Starting with the groundbreaking work of Bitran and Hax (1977), an enormous amount of literature has appeared describing hierarchical approaches to planning, scheduling, sequencing, and control problems that arise in manufacturing. The interested reader is referred to Kolisch (2001). Within the production planning and scheduling literature, very few papers address the problem of scheduling batches of jobs on multiple production lines with the goal of minimizing total setup costs (Allahverdi et al. 1999, 2008), and none of those pertain to the situation where a line can accommodate multiple batches or families at the same time. This generalization changes the problem significantly.

4 Solution methodology with example

For the real problem instances that we investigated the corresponding graph has 2606 nodes, an incredibly large number of constraints, and over six million variables. Therefore, finding optimal solutions with exact methods is not possible. In addition, the company required that a schedule be produced in 30 minutes or less. In pursuing a heuristic solution strategy, our goal was to develop a procedure that could find high quality solutions within this 30-minute time limit.

As such, we chose to develop a GRASP due to the success of using this procedure for the individual line sequencing component of the problem demonstrated in Monkman et al. (2005, 2008). In addition, the straightforward nature of a GRASP allows it to be easily modified to accommodate changes in the business environment and makes it easily explainable to the factory's production schedulers. For the origins of GRASP and a review of applications, see Feo and Bard (1989), Feo et al. (1991), and Feo and Resende (1995). In the last 20 years, GRASP has proven useful in a wide variety of applications including vehicle routing, sequencing and scheduling, telecommunications, and genetic mapping, to name a few.

Festa and Resende (2009a, 2009b) give a thorough bibliography of publications on GRASP through 2008 and categorize them by algorithm information and by application area. Two other papers demonstrating the breadth of applications using GRASP include Higgins et al. (2008) on environmental investment decision making and Hirsch et al. (2007) who analyze the relationship between drugs and adverse reactions. However, we are not aware of any research articles that apply a GRASP to a nonlinear optimization problem where the model is complex enough that feasibility cannot be maintained during initial solution construction.

4.1 GRASP outline

In general, a GRASP has two phases. In Phase 1, many feasible solutions are constructed by adding one element at a time to a partial solution in a greedy fashion. The idea is to maintain feasibility, except that instead of always adding the next lowest cost item, one is randomly selected from a short list of the next few most attractive items. A percentage of solutions with the best objective function values are saved from Phase 1 and put through Phase 2, which is designed to achieve local optimality using neighborhood search. The solution with the best objective function value at the end of Phase 2 is reported as the final solution.

The GRASP developed here generally follows this pattern but was altered slightly to deal with the nonlinear constraints (1j), which make it impossible in Phase 1 to reliably maintain feasibility. To overcome this difficulty, feasible solutions are built with respect to all constraints except (1i) and (1j), the latter also being troublesome to satisfy during construction. The result is checked for feasibility with respect to (1i) and (1j) and if not feasible but easy to fix, the necessary effort is applied; otherwise it is discarded. This happens before the end of Phase 1. In addition, a tie-breaking rule was developed in the case of multiple arcs having the same lowest cost when constructing the list of arcs to randomly choose from. A part commonality measure, *PCM*, was developed to give preference in the case of ties to arcs whose families in the node at the end of the arc have the most parts in common with each other. *PCM* is a static number calculated for each family node by taking the sum of the number of families in the node that require each part and dividing the result by the total number of parts required by at least one family in the node. The formula for node n is as follows,

$$PCM_n = \frac{\sum_{p \in I_n} \sum_{j \in G_n} f_{pj}}{|G_n|}$$

where I_n is the set of product families in node n, G_n is the set of all parts required by at least one family in node n, and parameter $f_{pj} = 1$ if product family p requires part j, 0 otherwise. Given that each node represents three families, the value of PCM_n will be between 1 and 3. The logic behind this tie-breaking rule is to give preference to nodes that potentially will have lower cost arcs leaving it since, when PCM_n is high, the cost of taking any family off line n will be relatively low.

The GRASP we designed required the specification of four parameters: *NA*, *NSOL*, *NSK*, and *MMV*. The first, *NA*, is the number of arcs in the restricted candidate list (*RCL*). One such arc is randomly chosen to be the next arc added to the partial solution. At each iteration, the *RCL* contains the lowest cost arcs among all the feasible arcs that could still be added to the solution. *NSOL* is the total number of feasible solutions to be generated in Phase 1 and *NSK* is the number of feasible solutions from Phase 1 that are kept and passed to Phase 2 for improvement. They are stored in a list called *NSK_List*. The fourth parameter, *MMV*, is the maximum number of missing visits that a solution can have to be considered repairable.

'Missing visits' refers to the number of times that a family needs to be in a node on a line's solution path to satisfy constraints (1j) minus the number of times that family currently appears in the line's path.

An outline of the full GRASP follows.

Phase 1

$i = 0$

WHILE $i < NSOL$ DO

Step 1. Empty all paths, initialize all variables except i and clear *NSK_List*.

Step 2. FOR each production line k DO
assign as first node on path p_k the families left on line k at end of previous shift.

Step 3. FOR ($j = 1$ to NA) DO
Find lowest cost over all eligible arcs; to be eligible, an arc must maintain feasibility with respect to constraints (1b), (1e), (1f), (1m).
IF there are multiple lowest-cost arcs THEN
Chose arc with lowest *PCM* value;
ELSE choose unique lowest cost arc.
Add chosen arc to *RCL* and put $j \leftarrow j + 1$.

Step 4. Randomly choose arc from *RCL* and add node at end of chosen arc to end of path for the correct line (i.e. the path that currently ends at node at origin of chosen arc)

Step 5. Check termination conditions: all lines assigned at least 2 high runners; all high runners assigned to at least 2 lines; all low runners assigned to 1 line.
IF terminating conditions not satisfied THEN
Go to Step 3.

Step 6. FOR each family f DO
IF f is a low runner THEN
Assign all jobs for family f to its assigned line;
ELSE split jobs for family f evenly across all lines to which it is assigned.

Step 7. IF solution does not satisfy constraints (1i) THEN
Discard solution and Go to Step 1.

Step 8. IF solution does not satisfy constraints (1j) THEN
IF missing number of node-visits $>$ *MMV* THEN
Discard solution and Go to Step 1.
FOR each line k that does not satisfy constraints DO
WHILE number of missing visits on path > 0 DO
IF no eligible nodes exists THEN
Discard solution and Go to Step 1.
ELSE For each node eligible to be inserted into or appended to path p_k (maintaining feasibility with respect to all other constraints), calculate the number of missing visits that would be gained and divide total incremental cost of adding node by the number of missing visits that would be gained to get cost-per-needed-visit.
Insert or append lowest cost-per-needed-visit node.

Step 9. Solution is now feasible; put $i \leftarrow i + 1$.
Calculate cost of new solution.
IF new solution's cost $<$ solution cost for any solution in *NSK_List* THEN

Replace higher cost solution in *NSK_List* with new solution.
ELSE discard solution.

Phase 2
FOR each solution i in *NSK_List* DO
FOR each line k DO
Step 1. For solution i's line k, calculate the cost reduction of swapping each pair of neighboring nodes while maintaining solution feasibility; save best cost reduction swap.
Step 2. For solution i's line k, calculate the cost reduction of removing each node; save best cost reduction node-removal while maintaining solution feasibility.
Step 3. IF either cost reduction from Steps 1 and 2 is positive THEN Perform highest cost reduction move from Steps 1 and 2 then Go to Step 1.
RETURN lowest cost solution in *NSK_List*

Step 2 enforces constraints (1c), (1d) and (1n), Step 3 enforces (1b), (1e), (1f), and (1m), Step 5 enforces constraints (1o) and (1p), Step 7 enforces (1i) and Step 8 enforces (1j). Constraints (1g) and (1h) are addressed implicitly in Step 6.

4.2 GRASP example

To illustrate, consider a simplified version of the production scheduling problem with five families and two lines with two chassis areas each. We denote the five families with the letters A, ..., E. This means that each node in V will include two families. The full network is shown in Fig. 3, where the depot, O, will be connected to the family node representing the parts left on the line from the previous shift. Recall that the graph is not completely connected because only one family can be changed during a setup.

Before we can start, we need to have the setup costs and the family demand data as well as a few parameter values. The node-to-node setup costs are given in Table 1, with the final column showing the *PCM* values for the nodes.

To keep the example simple we will produce one solution in Phase 1 and carry it over to Phase 2 for improvement, so $NSOL = 1$ and $NSK = 1$. We use the values 2 and 3 for parameters *NA* and *MMV*, respectively, and assume that families A and B are the high runners. Demand for families A through E is 2560, 1852, 493, 98, and 5, respectively, and $L^{up} = 4$, $L^{low} = 2$, and $L^{cap} = 2000$.

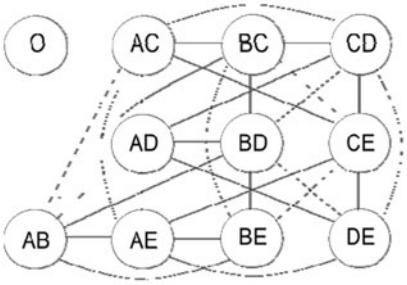


Fig. 3 Full network for example

Table 1 Arc costs and PCM values for example

	AB	AC	AD	AE	BC	BD	BE	CD	CE	DE	*PCM*
AB	–	47	66	42	36	66	33	–	–	–	1.00
AC	47	–	79	33	25	–	–	60	21	–	1.17
AD	66	79	–	74	–	28	–	33	–	29	1.01
AE	42	33	74	–	–	–	27	–	26	61	1.14
BC	36	25	–	–	–	82	35	63	32	–	1.07
BD	66	–	28	–	82	–	77	47	–	41	1.11
BE	33	–	–	27	35	77	–	–	39	64	1.08
CD	–	60	33	–	63	47	–	–	65	32	1.09
CE	–	21	–	26	32	–	39	65	–	71	1.36
DE	–	–	29	61	–	41	64	32	71	–	1.08

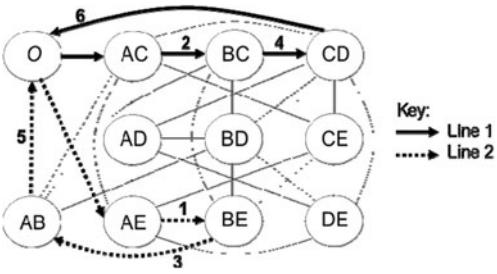


Fig. 4 Example problem initial infeasible solution

In Step 1, all the variables and arrays are initialized. In Step 2, we add arcs from the origin nodes for each line to the node representing families left on the line from the previous shift. We assume these are AC for line 1 and AE for line 2. In Step 3, we build *RCL* by searching for the two lowest cost eligible arcs that have their origin at the end of a line's path. The lowest cost arc leaving AC or AE is AC → CE at a cost of 21, but this is not eligible since using this arc would assign family E to line 1 and family E is a low runner that is already assigned to line 2. Arc AC → BC is the next lowest cost arc (cost = 25) and is eligible so it is added to *RCL*. The next lowest cost arc is AE → CE (cost = 26) but is not eligible since it would cause low-runner C to be assigned to two lines. The next arc is AE → BE with cost = 27 and is eligible so it is added to *RCL*.

At this point we have *NA* arcs on *RCL* so we move on to Step 4 where we randomly chose one, say, AE → BE. Being the first arc chosen, it is labeled 1 in the graph shown in Fig. 4. In Step 5, the terminating conditions are checked but they are not satisfied because, for example, the high runners are not each assigned to two lines. Returning to Step 3, *RCL* is constructed again, and arc AC → BC is chosen in Step 4. Iterating through Steps 3 and 4, we add arcs BE → AB and BC → CD. At this point the terminating conditions in Step 5 are satisfied and the arcs from the final node on each path back to the depot are implicitly added; line 1's path consists of $O \rightarrow$ AC → BC → CD → O and line 2's path is $O \rightarrow$ AE → BE → AB → O as shown in Fig. 4 with the arcs numbered in the order that they were added. Note that when constructing *RCL*, we did not encounter the situation where there were multiple eligible arcs with the same cost, so we did not need to use the *PCM* values.

In Step 6, the demand for each family gets assigned to the production lines. Families A and B are high runners so lines 1 and 2 will each have 1280 units of demand for family A and 926 units for family B. The other families are on one line each so their demand is assigned accordingly: line 1 gets 493 units of family C and 98 units of family D, and line 2 gets 5 units of family E. Line 1 has 2797 total units of demand and Line 2 has 2211 units. Therefore, each line has more than the required 2000 units so constraints (1i) in Step 7 are satisfied.

In Step 8, the constraints ensuring that each family will have enough time on the line need to be checked. For example, for family A on line 1, the left-hand side of constraint (1j) is $1/3$ and the right-hand side is $1280/2797$, which indicates a violation. Because the smallest integer needed in the numerator of the left-hand side that would satisfy the constraint is 2, and the value in the numerator is currently 1, family A has $2-1=1$ missing visit on line 1. Performing the same calculations for the other families, we find that line 1 has 2 missing visits and line 2 has 0 missing visits. Since the total number of 2 missing visits is less than *MMV*, we attempt to repair the solution.

Line 1 needs one more node-visit for both families A and B. The infeasible path of family nodes for line 1 is AC → BC → CD. Checking for eligible neighboring node swaps and node insertions we find only two possibilities that maintain feasibility: insert node AB after node AC or insert node BD after node BC. Inserting node AB will increase the solution cost by 58 but will satisfy two needed node-visits so the cost-per-needed-visit is 29. Inserting node BD will increase the cost by 66 and only satisfies one needed visit. Therefore the lowest cost move is to insert AB. The new path for line 1 is $O \rightarrow$ AC → AB → BC → CD → O. Rechecking constraints (1k) for the families on line 1 we see that all are now satisfied, as well as all the constraints. The cost is lower than the best cost in *NSK_List* (given that this list is initialized with very high values) so we save it in *NSK_List*. At this point we have generated *NSOL* feasible solutions so we move on to Phase 2. One thing to note is that node AB is now on both lines' solution paths.

In Phase 2, the lines associated with all solutions in *NSK_List* are run through the improvement procedure of node-swapping and node-deletion. The path for Line 1 is $O \rightarrow$ AC → AB → BC → CD → O. Examining possible node swaps in Step 1, we find that no node-swap is feasible since AC must come first and AB → CD is not allowed. Looking at node eliminations in Step 2, we find that none of the nodes can be removed. AC must remain since it must be first, AB and CD cannot be removed because constraints (1k) would be violated, and BC cannot be removed since AB → CD is not allowed. Therefore, no improvements are possible for Line 1.

The path for Line 2 is $O \rightarrow$ AE → BE → AB → O and has a cost of 60. In Step 1, we see that the only feasible node-swap is swapping nodes BE and AB. This will result in a new path cost of 75, so the best node-swap cost reduction is −15. Moving to Step 2, we examine possible node eliminations. Node AE cannot be removed since it must be first, so the first node elimination where feasibility can be maintained is the elimination of node BE. Removing node BE will give a new path cost of 42, yielding a cost reduction of 18. Removing node AB is also feasible, and will result in a new path cost of 27, with a cost reduction of 33. Moving to Step 3 we make the best cost saving move, which is to eliminate node AB for a cost reduction of 33. The new Line 2 path is $O \rightarrow$ AE → BE → O with a cost of 27. Returning to Step 1, we see that no node swaps are possible since AE must come first. And in Step 2 no node eliminations are possible since removing either node would violate constraints (1j). Since no more improvements are possible, Phase 2 is completed for this solution. All solutions in *NSK_List* have now gone through Phase 2 so the GRASP terminates with solution $O \rightarrow$ AC → AB → BC → CD → O and $O \rightarrow$ AE → BE → O, as shown in Fig. 5. The corresponding cost is 173.

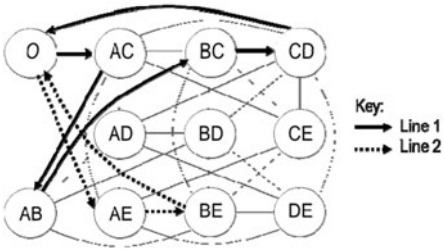


Fig. 5 Example problem final solution

5 Computational results

The GRASP was coded in C++ using the Eclipse integrated development environment (http://www.eclipse.org) and run on a 2.40 GHz desktop computer with 3.25 GB of RAM. Prior to the comparative testing, a set of experiments was performed to select the best parameter values for the GRASP. Given these values, the GRASP was compared to the MMB heuristic using actual factory data. The problem instances used in both the parameterization experiments and the comparisons all have 2600 family nodes, 26 product families, and 6 production lines. These results are analyzed and a data pre-processing routine, motivated by these results, is added to improve the performance of the GRASP.

The company has not authorized the publication of the family demand data but an overview of its characteristics may be helpful to understand additional difficulties in finding solutions. The family demand is highly variable among families. For any given shift, the demand for some families is in the single digits while for others it is in the thousands. In addition, demand for any single family across shifts is highly variable. For the high runner families the coefficient of variation (CV) across shifts ranged from 0.7 to over 1.3 while the CV for low runner families ranged as high as 13. This variability causes difficulties in Steps 7 and 8 of the GRASP resulting in many solutions being discarded as infeasible. Further discussion of this issue is provided in Sect. 5.2.

5.1 GRASP parameter selection

The values of the four parameters used in Phase 1 affect both the runtime and quality of the solutions. Since we are aiming for runtimes of 30 minutes or less, and the objective is to reduce setup costs, we needed to gain a clear understanding of what the best settings would be. To do this, we performed experiments to investigate the performance of the GRASP by varying the levels of each parameter: *NA* (number of arcs on the *RCL*), *MMV* (maximum missing visits for a solution to be considered 'repairable'), *NSOL* (number of feasible solutions to generate in Phase 1), and *NSK* (number of solutions from Phase 1 to keep for improvement in Phase 2). Due to the fact that we had a limited number of data sets of real family demand from the factory, we fitted probability distributions to each family's demand and then used these distributions to generate demand data for the parameterization experiments.

For the experimental design, we chose to use NOLH, which allowed us to efficiently vary each of the four factors (the four parameters in the GRASP) over a wide range without requiring an exponential number of runs. For example, in our first experiment, we wanted to investigate a range of 2 to 40 for the parameter *NA*. Varying this factor alone at every level would result in 39 scenarios. When combined with varying the *MMV* parameter factor from 2 to 20, we would have $39 \times 19 = 741$ scenarios, and varying the other factors in this way would continue this drastic increase. On the other hand, if we just varied each factor by

Table 2 Factor ranges and scenarios used in Experiment 1

	NA	*MMV*	*NSOL*	*PSK*
Lowest value	2	2	10000	0.01
Highest value	40	20	50000	1.00
Scenario 1	14	20	42500	0.38
Scenario 2	4	7	45000	0.57
Scenario 3	7	10	12500	0.26
Scenario 4	9	13	22500	1.0
Scenario 5	31	19	27500	0.13
Scenario 6	40	8	25000	0.81
Scenario 7	26	5	50000	0.32
Scenario 8	23	18	40000	0.94
Scenario 9	21	11	30000	0.51
Scenario 10	28	2	17500	0.63
Scenario 11	38	16	15000	0.44
Scenario 12	35	12	47500	0.75
Scenario 13	33	9	37500	0.01
Scenario 14	12	3	32500	0.88
Scenario 15	2	14	35000	0.20
Scenario 16	16	17	10000	0.69
Scenario 17	19	4	20000	0.07

using its lowest and highest values, we would only have $2^4 = 16$ scenarios, but we would have no idea of the effects of these factors in the middle of the range.

The NOLH design can be used to select varying factor levels throughout their desired range in such a way that they have essentially no correlation with each other while keeping the number of scenarios very low. With four factors, we were able to use the NOLH design with only 17 scenarios and get a good representation of parameters in these ranges. Table 2 gives the factor ranges and factor settings for each scenario used in our first parameterization experiment, and shows how effectively the NOLH design can cover the ranges with only a few scenarios. More information on NOLH can be found in Cioppa (2002), Cioppa and Lucas (2007) and Kleijnen et al. (2005). The worksheet used to quickly calculate NOLH designs was developed by Susan Sanchez and can be found at http://harvest.nps.edu.

In setting up the experiments, one change was needed to accommodate NOLH design requirements related to parameter independence. Because it is not possible to save more solutions for improvement than are generated, we could not vary *NSK* independently of *NSOL*. To resolve this issue, *NSK* was replaced with the percentage of solutions kept (*PSK*) for improvement in Phase 2. *NSK* was then calculated from *NSOL* and *PSK* for use in the GRASP.

The variability in the output of our GRASP comes from two sources. The first is the different demand levels for each product family that are part of the input. The second is the seed value used for the random number generator within the procedure. We chose to keep the random number seed constant across runs and only vary the demand levels for the families. To represent the variability in family demand data, we used the fitted probability distributions to generate 10 different values for each family so we could run 10 replications of each scenario in the experimental design.

Table 3 Factor ranges used in Experiment 2

	NA	*MMV*	*NSOL*	*PSK*
Lowest value	25	5	100000	0.01
Highest value	50	20	400000	1.00

Two experiments were designed and run to test varying parameter levels. The first experiment tested the parameter factor levels shown in the top section of Table 2 and included the 17 scenarios shown in the bottom section of the table. Each scenario was run on the 10 different sets of demand data, for a total of 170 runs. The JMP software, version 7.0.2, was used to analyze the results of each experiment. The results of the first experiment were averaged across the 10 replications for each scenario. A regression model was fitted to the corresponding 17 data points using stepwise regression and the prediction profiler was used to ascertain the general behavior of the solution cost and solution time as each factor was varied. The results indicated that the *NA* factor was predicted to have much more influence on the solution costs with values decreasing as the factor values increased from the lowest end of the range and then leveled off somewhat with a slight dip in the 30 to 40 range. The other three factors had very small effects on predicted solution costs with costs slightly increasing in *MMV* and slightly decreasing in both *NSOL* and *PSK*. With regard to solution time, the predicted time decreased sharply for low values of *NA* and then leveled off, but showed a decrease again in the largest values of *NA*. The predicted time also decreased sharply for low values of MMV and then rose and fell slightly as MMV increased. Solution time was linearly increasing in *NSOL*.

For Experiment 2, the factor ranges were changed for *NA, MMV* and *NSOL*. The range for *NA* was changed to 25–50 to focus on the range where the predicted costs leveled off and the upper end of the range was extended. The low end of the *MMV* range was increased to prevent the exceptionally long solution times for low values of *MMV* seen in Experiment 1. And, since solution times were generally low in Experiment 1 and it seemed logical that generating more solutions should increase the chance of finding a lower cost, a range of larger values of *NSOL* was used. The factor ranges for Experiment 2 are given in Table 3.

The data from Experiment 2 was averaged within scenarios, a regression model was fitted to it, and the prediction profilers were examined for solution cost and time. The results showed that all four variables influenced the solution costs. For *NA*, the predicted costs decreased and then increased as *NA* increased, with a low at $NA = 35$. This value was robust in that it remained the low value regardless of the values of the other three parameters. *MMV*, *NSOL*, and *PSK* were predicted to be linearly related to the solution cost but the direction of each relationship depended on the values of the other two of these three parameters. Analysis of these relationships, along with consideration of predicted solution time using a regression model constructed, resulted in the selection of the following values to minimize solution costs with the expectation of not exceeding the 30-minute limit: $NA = 35$, $MMV = 5$, $NSOL = 330000$, and $PSK = 1.0$.

5.2 GRASP performance on factory data

Using these values, the GRASP was run on 10 instances derived from the raw factory data and the results compared to those obtained with the MMB heuristic. In this section, we highlight the performance of the GRASP in Table 4 and report Phase 1 and Phase 2 statistics in Table 5. The comparisons with the MMB heuristic are presented in the next section.

One of the interesting characteristics of the production sequencing problem investigated here is the fact that some of the constraints are nonlinear, and by design, the initial solutions constructed by the GRASP are not necessarily feasible. The second column in Table 4 shows the total number of initial solutions generated in Phase 1 for each scenario. As can be seen, in some cases, many initial solutions were infeasible and ultimately had to be discarded. This greatly influenced the runtimes for Phase 1, shown in column 4. In Phase 2, the same number of solutions was always processed, so very little variation is seen in runtimes (column 7). In addition, we can see that although almost all solutions were improved in Phase 2 (column 8), the Phase 2 runtimes were much smaller than Phase 1 runtimes. Columns 5 and 9 give the values of the solution costs after Phase 1 and Phase 2, respectively, and the last column shows that, in all scenarios, Phase 2 provided a significant improvement averaging 8.0 %.

Table 5 provides further information on why so many solutions are discarded during Phase 1. The second column indicates the number of solutions discarded because no more feasible arcs could be found in Step 3. Column 3 gives the number of solutions discarded in Step 7 due to too little demand being assigned to at least one line. Column 4 shows the number of solutions that were discarded because the number of missing node-visits was greater than *MMV*. The results of discarding only one solution over all ten scenarios, as seen in this column, indicate that higher values of *MMV* would not be expected to improve the results. The fifth column shows how many solutions were discarded in Step 8 due to the procedure being unable to fix the solution to make the number of node-visits satisfy constraints (1j).

To summarize, the last two columns show the total number of solutions discarded and the overall percent of solutions that were discarded. Evidently, the nonlinear nature of the problem coupled with the wide variation in family demand greatly restrict the usability of an exceedingly large number of solutions found in Phase 1.

5.3 Comparison of production sequencing heuristics

Given the results in Table 4, we now compare the final best solution costs for the GRASP to the best solution costs found with MMB. In addition to using the same demand data, each heuristic was initialized with the same family nodes for each production line, representing the families left on the line from the previous shift. To ensure these initial families were realistic, they were chosen by using the model developed by Loveland et al. (2007) to assign families to lines. From the given assignments, the initial subset of families to represent those left on the line were randomly selected for each line.

Table 6 reports the setup costs associated with the final sequencing solutions produced by each procedure. The fourth column gives the percent difference between the GRASP and the MMB heuristic. In 5 out of 10 cases, the GRASP outperformed the MMB heuristic (Scenarios 1, 3, 5, 9, and 10), with an improvement ranging from 2.11 % to 19.34 %. However, the MMB heuristic still performed much better in four of the 10 cases (Scenarios 4, 6, 7, and 8), which caused the overall average improvement of the GRASP to be −12.92 %. From the statistics in the table, we see that there is a great deal of variability in the results using either heuristic, and we see that the MMB heuristic gives the most variable results, generating the solutions with the lowest and the highest overall costs.

To better understand these results, we investigated the different demand data that was used in each scenario. We found no identifiable difference that could explain why one procedure performed better or worse for certain scenarios. Noting that most of the results fall in the general range of 600–800, with the exceptions being the four scenarios where MMB performed much better, we then focused on the solutions produced by each heuristic. Our

Table 4 GRASP performance statistics using factory demand data

Scenario	Number initial solutions generated	Number feasible solutions generated	Phase 1 time (sec)	Phase 1 best solution	Number solutions kept from Phase 1	Phase 2 time (sec)	Number solutions improved in Phase 2	Phase 2 best solution	Improvement from Phase 1 best to Phase 2 best (%)
1	461128	330000	585	718	330000	17	329843	648	9.75
2	460686	330000	599	729	330000	17	329750	691	5.21
3	1603886	330000	1574	733	330000	17	329826	697	4.91
4	1286674	330000	1396	797	330000	17	329847	709	11.04
5	1562872	330000	1583	733	330000	18	329859	717	2.18
6	855833	330000	945	765	330000	17	329777	704	7.97
7	1250865	330000	1268	719	330000	18	329855	659	8.34
8	1464707	330000	1524	791	330000	18	329823	719	9.10
9	471682	330000	564	719	330000	17	329818	634	11.82
10	466505	330000	630	738	330000	17	329854	667	9.62

Table 5 GRASP discarded solutions in Phase 1

Scenario	No more feasible arcs available	Too little demand on at least 1 line	Too many missing node visits	Unfixable missing node visits	Total discarded	Percent discarded
1	70738	13513	0	46877	131128	28.44
2	70671	0	0	60015	130686	28.37
3	246388	971015	0	56483	1273886	79.42
4	197220	611665	1	147788	956674	74.35
5	239976	904949	0	87947	1232872	78.89
6	130834	334799	0	60200	525833	61.44
7	191641	697010	0	32214	920865	73.62
8	224884	809861	0	99962	1134707	77.47
9	72397	43985	0	25300	141682	30.04
10	71591	0	0	64914	136505	29.26

Table 6 Comparison of setup costs

Scenario	MMB heuristic cost	GRASP cost	GRASP total time (sec)	Cost improvement, GRASP vs. MMB (%)
1	662	648	602	2.11
2	687	691	616	−0.58
3	786	697	1591	11.32
4	453	709	1413	−56.51
5	766	717	1601	6.40
6	484	704	962	−45.45
7	461	659	1286	−42.95
8	540	719	1542	−33.15
9	786	634	581	19.34
10	743	667	647	10.23
Average				−12.92

investigation showed that MMB solutions for scenarios 4, 6, 7, and 8 each had several individual lines with unusually low setup costs whereas the other solutions, across both procedures, usually had only one or zero lines with unusually low setup costs. However, the lines without unusually low costs in these 4 best solutions appeared to have costs similar to line costs in all other solutions. So the primary difference seemed to be in these few unusually low cost lines. Investigating these lines, we saw that the part commonality of the families assigned to these lines was very high. In addition, the demand distribution for the families on the line allowed one family with a larger number of parts to remain on the line for the full shift so the other families very often had most of their required parts already on the line, drastically reducing the setup costs.

Using this information, we then examined the two procedures to see why MMB was able to find these low-cost lines but the GRASP wasn't. We observed that one advantage exhibited by the MMB heuristic is that it explicitly considers the number of parts assigned to a line in the first step of the decomposed solution procedure. This directly contributes to

the conditions found in the four best solutions. In the GRASP, however, the number of parts being assigned to a line is not explicitly addressed. For each iteration of adding a node in the solution, only the setup cost for the next arc under investigation is considered, which can allow a family assigned to a line in earlier nodes on a path to have very different part requirements than a family assigned to the line more than a node or two later in the path. Moreover, the explicit consideration of part requirements in the GRASP is only included in the tie-breaking rule when adding arcs to the *RCL*.

From this investigation, it appears that the non-traditional nature of the sequence-dependant setup costs in which family assignments early in a line's path can significantly affect setup costs later in a line's path due to part commonality issues, is a major contributing factor to the inability of the GRASP to find better solutions in some cases. The fact that MMB can sometimes find much lower cost solutions indicates that there could potentially be significant savings achieved if the GRASP could more explicitly consider the part requirements of the families assigned to each line. Since the families and their part requirements rarely change, we determined that data pre-processing based on family part requirements could potentially improve the performance of the GRASP, while being very practical in the real-world setting.

5.4 Data pre-processing

The data pre-processing undertaken consisted of using the established *PCM* metric to determine the pairwise *PCM* for all possible family pairs. A list of the pairs of families with the highest *PCM* values was developed (good pairs), as well as a list of the pairs with the lowest *PCM* values (bad pairs).

We then redesigned the GRASP to read in the lists of good and bad pairs and to use this information when building solutions. More specifically, assignments of the two families in each bad pair to the same line were prohibited, and assignments of the two families in each good pair to the same line were accepted whenever possible, during solution construction. The following additions were made to the GRASP as it was described in Sect. 4:

- Following Phase 1, Step 2:
  - For each family assigned to the line: if it appears in the bad pairs list, add that family's bad pair(s) to the line's bad families list if it isn't already there; then
  - For each family assigned to the line: if it appears in the good pairs list, add that family's good pair(s) to the line if it is not already on the line AND it is not on the line's bad families list AND other conditions for adding families to lines is met (so constraints (1m), (1o), and (1p) can be maintained)
- In Phase 1, Step 3: an arc is also not eligible if the new family that would be added to the line by adding the node at the end of the arc is on the line's bad families list
- Following Phase 1, Step 4:
  - For the new family added to a line: if it appears in the bad pairs list, add that family's bad pair(s) to the line's bad families list if it isn't already there; then
  - For the new family added to a line: if it appears in the good pairs list, add that family's good pair(s) to the line if it is not already on the line AND it is not on the line's bad families list AND other conditions for adding families to lines is met (so constraints (1m), (1o), and (1p) can be maintained)

With the incorporation of the use of good pairs and bad pairs, there are two additional parameters that must be decided: the number of good pairs to use (*NGP*) and the number of bad pairs to use (*NBP*). In addition, the incorporation of the pre-processing information

Table 7 Factor ranges used in New Experiment

	NA	*MMV*	*NSOL*	*PSK*	*NGP*	*NBP*
Lowest value	15	10	50000	1.00	0	0
Highest value	50	20	100000	1.00	15	17

caused many more solutions to be discarded during Phase 1 so the number of solutions generated needed to be reduced significantly, and the effects of the other parameters was no longer clear. Therefore, a new NOLH-designed experiment was run to determine the best parameter values to use. Since the prior experiments had indicated that higher values of the % kept parameter had a negligible effect on solution times and could not yield worse solutions, we chose to retain all solutions.

For the new parameterization experiment, a bad pairs list was developed using all pairs of families with 2 % or fewer parts in common (21 unique pairs), rank ordered by increasing *PCM* values. And the good pairs list contained all pairs with 30 % or more pairs in common (up to the highest percent in common of 60 %, 27 unique pairs), rank ordered by decreasing *PCM*. However, preliminary runs showed that using large numbers of bad and good pairs yielded no feasible solutions. In addition, low values of *NA* and *MMV* were seen to cause excessive solution times, so their low values were also adjusted. Therefore the new parameterization experiment used the ranges shown in Table 7.

The results of the new parameterization experiment showed that higher *NGP* and *NA* values most significantly contributed to lower costs, and avoiding the highest *MMV*, *NGP*, and *NBP* values, and using moderate *NA* values reduced the solution times. In addition, moderate *MMV* values and *NBP* values also contributed to smaller improvements in cost. A higher number of solutions did not have as much impact as expected, but did contribute somewhat to lower costs and higher solution times. However, with the appropriate selection of the other parameters, it was still anticipated that the GRASP would be able to solve within 30 minutes for a variety of demand scenarios even when finding 100,000 solutions. This analysis resulted in the selection of the following values to minimize solution costs with the expectation of not exceeding the 30-minute limit: $NA = 40$, $MMV = 13$, $NSOL = 100000$, $PSK = 1.0$, $NGP = 11$, and $NBP = 6$.

Using the GRASP with the incorporated data pre-processing yielded the results shown in Table 8. Clearly the data pre-processing allows the GRASP to be able to find many lower cost lines that it couldn't find without this additional information; it now performs better than the MMB heuristic in 7 out of 10 cases, and by more than 17 % on average over all 10 scenarios. At best, the GRASP with pre-processing performs over 38 % better, and at worst performs 12.6 % worse. A paired t-test on the difference in the results between the MMB heuristic and the GRASP with pre-processing (columns 2 and 3 of Table 8) indicates a highly significant negative average difference (p-value = 0.0075).

These results confirm that explicit consideration of a basic element of the non-traditional setup costs, i.e., part commonalities, dramatically improves the performance of the GRASP. Upon reflection, this appears logical since one characteristic of GRASP is that it is myopic, considering only the next node to be added on any path. In the case of traditional sequence-dependant setup costs, this characteristic is not a limitation but it is here. Our results show that the addition of pre-processing to explicitly consider the underlying cause of the non-traditional nature of the setup costs, with appropriate adjustments to the GRASP, significantly helps GRASP overcome this limitation.

Table 8 Comparison of setup costs with data pre-processing

Scenario	MMB heuristic cost	GRASP with preprocessing cost	GRASP with preprocessing total time (sec)	Cost improvement, GRASP with preprocessing vs. MMB (%)
1	662	481	109	27.34
2	687	484	114	29.55
3	786	531	345	32.44
4	453	493	525	−8.83
5	766	509	373	33.55
6	484	545	439	−12.60
7	461	482	255	−4.56
8	540	516	714	4.44
9	786	484	105	38.42
10	743	489	114	34.19
Average				17.39

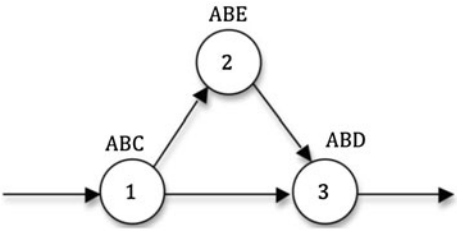
6 Summary and conclusions

In this paper, we described and modeled a real-world production scheduling problem with non-traditional sequence-dependant setup costs and several additional complications. To the best of our knowledge, this problem has not been studied before without being decomposed. We then presented a GRASP to solve the resulting nonlinear IP and demonstrated the use of NOLH designed experiments to aid in selecting the best parameter settings for the procedure. The initial results showed that the GRASP was able to outperform the existing heuristic in some cases, and had less variability in the results, but did not outperform the existing heuristic on average. A pre-processing step was then added, which dramatically improved the results, yielding an average improvement of over 17 % compared to the MMB heuristic. With pre-processing, the average improvement in the results of the GRASP over the MMB heuristic was shown to be statistically significant.

Our research demonstrates that the myopic nature of GRASP becomes a limitation in problems where sequence-dependant setup costs are non-traditional in that they depend on the decisions made earlier along the path leading up to the current node and not just on the state represented by the current node. In a larger context, this implies that the dynamic programming principle of optimality does not hold for our problem. We also demonstrate that this limitation can be significantly mitigated by explicitly considering the underlying source of the non-traditional nature of the setup costs through the addition of pre-processing.

In light of the performance of the new GRASP, at least two directions for future research offer the possibility for further improvement. The first would be to partially redesign the GRASP to include more complex adaptive mechanisms in Phase 1, as well as investigating more advanced exchange procedures such as ejection chains. The second would be to explore the use exact methods such as branch and price in conjunction with heuristics for generating columns.

Fig. 6 Three-node subgraph



Appendix: Triangle inequality proof

Using the same notation as in Sect. 3, let $G = (V, A)$ be an undirected graph representing an instance of the assignment and sequencing components of the full scheduling problem. Recall that the nodes in V correspond to each possible set of families that can be on a single line at the same time and the arcs in A correspond to feasible transitions between nodes. Two nodes i and j are connected if and only if they differ by a single family. The arc cost c_{ij} is equivalent to the number of parts that must be placed on the line plus those that must be removed when some family in node i is removed and the new family is added, where the new family is the family in node j that is not already on the line. Parts not needed for any family in node j must be removed implying the symmetric relationship $c_{ij} = c_{ji}$.

Also note that the node set V does not contain the depot. The constraints that define the problem restrict the path of any line to only go through the depot node O once, and the arcs leaving the depot are uniquely predetermined by the families on lines at the beginning of the shift. Therefore, it is not possible for node O to appear at an intermediate point along a path which prevents the use of a path through node O to obtain a lower cost than the cost of a path whose only difference is the exclusion of node O. The triangle inequality (TI) then only needs to hold for graph G.

Proposition 1 *The triangle inequality with respect to the cost matrix* $\mathbf{c} = (c_{ij})$ *holds for the graph* $G = (V, A)$.

Proof Without loss of generality, consider the three-node subgraph in Fig. 6 for the five families A, B, C, D, E. We need to show that $c_{13} \leq c_{12} + c_{23}$ (the other cases in the subgraph admit identical arguments).

The arc cost c_{ij} is the sum of the setup costs for each individual part, where the setup cost for part p is c_{ij}^p. It is therefore sufficient to prove that

$$c_{13}^1 + c_{13}^2 + \cdots + c_{13}^n \leq c_{12}^1 + c_{12}^2 + \cdots + c_{12}^n + c_{23}^1 + c_{23}^2 + \cdots + c_{23}^n \tag{2}$$

is true for every positive integer n.

For $n = 1$, which is the case where a single part contributes to the setup costs:

For the TI to be violated for two paths $1 \to 3$ and $1 \to 2 \to 3$, we must have $c_{13}^1 > c_{12}^1 + c_{23}^1$ which is only possible if $c_{13}^1 = 1$ and $c_{12}^1 = c_{23}^1 = 0$. For any single part, there are two possible cases that can result in $c_{13}^1 = 1$: either family C needs the part and family D does not, or family D needs the part and family C does not. In addition, we need to consider whether or not family E requires the part which, in combination with the first two cases, gives us four cases.

Case 1: Families A, B, D and E do not require the part and family C does require the part.

Since one of the families in node 1 requires the part, and no family in node 2 requires the part, the part must be removed from the line on arc $1 \rightarrow 2$ so $c_{12}^1 = 1$ and the TI is not violated.

Case 2: Families A, B and D do not require the part and families C and E do require the part.

In this case families in both nodes 1 and 2 require the part so $c_{12}^1 = 0$. However, none of the families in node 3 require the part so the part must be removed on arc $2 \rightarrow 3$, so $c_{23}^1 = 1$ and the TI is not violated.

Case 3: Families A, B, C and E do not require the part and family D does require the part.

In this case none of the families in nodes 1 or 2 require the part so $c_{12}^1 = 0$. However, family D does require the part so the part must be added on arc $2 \rightarrow 3$, so $c_{23}^1 = 1$ and the TI is not violated.

Case 4: Families A, B, and C do not require the part and families D and E do require the part.

Since none of the families in 1 require the part, but a family in 2 requires the part, the part does need to be added on arc $1 \rightarrow 2$ so $c_{12}^1 = 1$ and the TI is not violated.

Therefore there is no possible case for which the TI can be violated when $n = 1$.

Now we assume the TI holds for k:

$$c_{13}^1 + c_{13}^2 + \cdots + c_{13}^k \leq c_{12}^1 + c_{12}^2 + \cdots + c_{12}^k + c_{23}^1 + c_{23}^2 + \cdots + c_{23}^k \tag{3}$$

where k is a positive integer.

So for the TI to hold we need to prove that

$$c_{13}^1 + c_{13}^2 + \cdots + c_{13}^k + c_{13}^{k+1} \leq c_{12}^1 + c_{12}^2 + \cdots + c_{12}^k + c_{12}^{k+1} + c_{23}^1 + c_{23}^2 + \cdots + c_{23}^k + c_{23}^{k+1} \tag{4}$$

In light of (3), the inequality in (4) reduces to $c_{13}^{k+1} \leq c_{12}^{k+1} + c_{23}^{k+1}$ which we now must show is true. However, this reduced inequality is equivalent to the case where $n = 1$, which we have shown above to be true.

Hence, by the principle of mathematical induction, (2) is true for every positive integer n and therefore the TI holds for G. □

References

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup consideration. *OMEGA The International Journal of Management Science*, *27*(2), 219–239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985–1032.

Bard, J. F. (1997). An analysis of a rail car unloading area for a consumer products manufacturer. *Journal of the Operational Research Society*, *48*(9), 873–883.

Bitran, G. R., & Hax, A. C. (1977). On the design of hierarchical production planning systems. *Decision Sciences*, *8*(1), 25–53.

Cioppa, T. M. (2002). *Efficient nearly orthogonal and space-filling designs for high-dimensional complex models*. Unpublished doctoral dissertation, Naval Postgraduate School, Monterey, CA.

Cioppa, T. M., & Lucas, T. W. (2007). Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, *49*(1), 45–55.

Feo, T. A., & Bard, J. F. (1989). Flight scheduling and maintenance base planning. *Management Science*, *35*(12), 1415–1432.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–134.

Feo, T. A., Venkatraman, K., & Bard, J. F. (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, *18*(8), 635–643.

Festa, P., & Resende, M. G. C. (2009a). An annotated bibliography of GRASP—Part I: algorithms. *International Transactions in Operational Research*, *16*, 1–24.

Festa, P., & Resende, M. G. C. (2009b). An annotated bibliography of GRASP—Part II: applications. *International Transactions in Operational Research*, *16*, 131–172.

Higgins, A. J., Hajkowicza, S., & Buib, E. (2008). A multi-objective model for environmental investment decision making. *Computers & Operations Research*, *35*(1), 253–266.

Hirsch, M. J., Meneses, C. N., Pardalos, P. M., Ragle, M., & Resende, M. G. C. (2007). A continuous GRASP to determine the relationship between drugs and adverse reactions. *Data Mining, Systems Analysis and Optimization in Biomedicine*, *953*, 106–121.

Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., & Cioppa, T. M. (2005). State-of-the-art review: a user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, *17*(3), 263–289.

Kolisch, R. (2001). *Make-to-order assembly management*. Berlin: Springer.

Laporte, G., Riera-Ledesma, J., & Salazar-González, J. J. (2003). A branch-and-cut algorithm for the undirected traveling purchaser problem. *Operations Research*, *51*(6), 940–951.

Loveland, J. L., Monkman, S. K., & Morrice, D. J. (2007). Dell uses new production scheduling heuristics to accommodate increased product variety. *Interfaces*, *37*(3), 209–219.

Mavridou, T., Pardalos, P. M., Pitsoulis, L. S., & Resende, M. G. C. (1998). A GRASP for the biquadratic assignment problem. *European Journal of Operational Research*, *105*, 613–621.

Monkman, S. K., Morrice, D. J., & Bard, J. F. (2005). Scheduling product families in a high volume, flexible, assemble-to-order factory. In G. Kendall, L. Lei & M. Pinedo (Eds.), *Proceedings of the 2nd multidisciplinary international conference on scheduling: theory & applications* (pp. 394–395).

Monkman, S. K., Morrice, D. J., & Bard, J. F. (2008). A production scheduling heuristic for an electronics manufacturer with sequence dependent set-up costs. *European Journal of Operational Research*, *187*(3), 1100–1114.

Pinto, J. M., & Grossmann, I. E. (1998). Assignment and sequencing models for the scheduling of process systems. *Annals of Operations Research*, *81*, 433–466.

Shen, Q., Chen, H., & Chu F. (2008). Model and algorithm for an inventory routing problem in crude oil transportation. *Journal of Advanced Manufacturing Systems*, *7*(2), 297–301.