Fulfilling Customer Orders for Steel Plates From Existing Inventory

Peter A. Huegler
Kutztown University

Joseph C. Hartman
Lehigh University

Report No. 04T-005

FULFILLING CUSTOMER ORDERS FOR STEEL PLATES FROM EXISTING INVENTORY

Peter A. Huegler
Department of Mathematics and Computer Science, Kutztown University, Kutztown, PA

Joseph C. Hartman
Industrial and Systems Engineering Department, Lehigh University, Bethlehem, PA

This paper investigates the steel plate order fulfillment problem from existing inventory, which is a generalization of one-dimensional cutting and packing problems. This problem is an integral part of the day-to-day operations of steel plate manufacturing. A 0-1 integer programming formulation is presented and two heuristics, based on single order formulations, are presented to quickly generate lower and upper bounds. The bounds are improved using Lagrangian Relaxation and Subgradient Optimization. The lower bound improvements reduce the gap over 30% and the upper bound improvements reduce the gap another 35%. The upper bound improvements represent an actual savings over the current solution method of 2%, or $4 million over a three-week period, for 17 real-world data sets analyzed. An economic interpretation of the Lagrange multipliers is also discussed and shown to be useful for handling rush orders.

1. INTRODUCTION

The production of steel plates, which are used in many applications, including bridges, deep-sea oilrigs, and construction equipment, at an integrated mill begins with the production of slabs from liquid steel (processed from iron ore) on a continuous caster. These slabs represent semi-finished inventory, as they are burned, heated, rolled, and sheared to fill customer orders for plates, as shown in Figure 1, which specify required dimensions, quantities, steel properties, and due dates. As slabs are produced in batches of 300 tons or more and customer orders are generally for 20-40 tons, it is inevitable that excess slabs will be produced and added to the slab inventory. Thus, an important problem faced in the industry is the assignment of slabs in inventory to customer orders. These decisions are important because they impact all aspects of

the planning process from slab production, plate mill productivity, process yields, inventory management, and customer service.

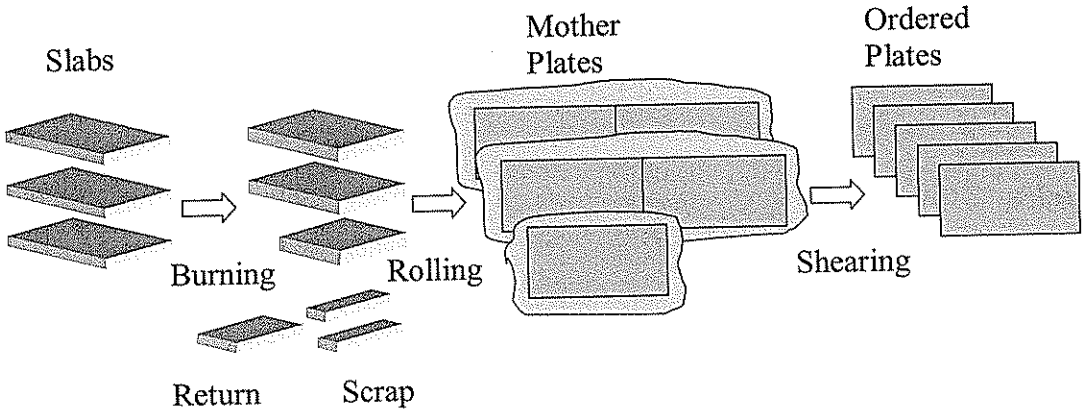

Figure 1 – Plate Production Process from Semi-Finished Slab Inventory

This paper focuses on the problem of assigning inventory slabs to customer orders. The burning step (and subsequent order fulfillment problem) appears as a classical one-dimensional cutting stock problem (1CSP) in the slab length, as slabs can only be made shorter for quality reasons. However, there are a number of differences between the order fulfillment problem and the classical 1CSP, namely (1) slab inventory is heterogeneous, varying by weight and steel properties; (2) only feasible slabs can be used to fulfill an order (slabs are feasible if the slab properties meet or exceed the orders properties) and slabs that exceed an order's properties can be used at an additional cost; (3) unused slabs which meet minimum size requirements may be returned to inventory; (4) economies of scale exist such that the production of multiple plates from a single slab leads to a greater productivity and a reduction in waste (this can be seen in Figure 1, as "end scraps" are eliminated with multiple plate production); (5) the objective function in the assignment problem includes a variety of cost considerations, as noted in Table 1.

Table 1 – Order Fulfillment Costs

Cost	Description
Slab Cost	Cost of the raw materials (slab) per ton. Slabs with better properties cost more than slabs with lesser properties.
Rolling Cost	Productivity costs. Cost to roll a slab into a mother plate.
Holding Cost	Cost of holding one ton of steel in the inventory for one time period.
Burning Cost	Cost to burn a slab into two pieces. This includes fixed setup cost and associated yield loss.
Customer Service Cost	Cost of not completing an order in time.
New Production Cost	Cost of having to order production from the caster (and not filling order from inventory).

We define the steel plate order fulfillment problem (SPOF) as follows: A set of slabs exists in inventory defined by size, weight and steel properties. Customer orders to be filled over a time horizon are defined by the number, dimension, properties and due dates of the steel plates. Orders that cannot be fulfilled through inventory are filled through new production at an additional cost. The goal is to fulfill the customer orders at minimum cost.

We model SPOF using 0-1 integer programming formulation and solve it using Lagrangian Relaxation and Subgradient Optimization. Two greedy heuristics based on single order formulations are presented to quickly generate lower and upper bounds, which are shown to be quite effective. The greedy upper bound heuristic is similar to the method used at the Burns Harbor mill of the Bethlehem Steel Corporation before the company declared bankruptcy. We present results on 17 real-world data sets, ranging in size from 1,500 to 5,000 orders and 4,000 to 11,000 slabs. Lagrangian Relaxation and Subgradient Optimization are used to improve the lower and upper bounds. The lower bound improvements reduce the gap by over 30%. The upper bound improvements reduce the gap over 34%. The upper bound improvements represent a savings of 2% compared to the greedy upper bound heuristic. Additionally, an economic

interpretation of the Lagrange multipliers is discussed and shown to be useful for handling rush orders. Unfortunately, the improvements remain unimplemented due to the bankruptcy of Bethlehem Steel and subsequent purchase of its assets by International Steel Group.

The paper is organized as follows: Section 2 reviews the relevant literature. Section 3 contains the 0-1 integer programming formulation of the SPOF. Section 4 discusses the single order problem, which is the basis of the bound and improvement methods. Section 5 presents two greedy heuristics for generating lower and upper bounds. Section 6 discusses the Lagrangian Relaxation and Subgradient Optimization bound improvements. Section 7 presents the results and Section 8 discusses a using the generated Lagrangian multipliers to handle rush orders.

2. LITERATURE REVIEW

There are a number of problems that arise in steel production that have similarities to SPOF. One similar problem is the steel coil order fulfillment problem, where slabs rolled into steel coils on a hot strip mill. This problem differs as customers generally place orders according to weight. Cohen, et al. (1984) report on a system that applies slabs to coil orders using expert systems. Kalagnanam, et al.(2000) formulate the coil order fulfillment problem as a bicriteria multiple knapsack problem with color constraints. The color constraints address the property requirements. A transportation formulation is presented in Vasko, et al. (1994) for the coil order fulfillment problem. This formulation assigns the orders to the sources and the slabs to the destinations. For a recent review of hot strip mill planning and scheduling methods, refer to Tang, et al. (2001b).

An additional problem in coil production is packing customer coil widths into production widths. Customers generally order coils in widths that are narrower than the maximum possible

production width. To increase productivity, several customer orders are packed into master coils (Haessler (1978), Haessler and Vonderembse (1979), Vasko, et al. (1992), Vonderembse (1995), and Vonderembse and Haessler (1982a, 1982b)).

One characteristic of the order fulfillment problem is the generalization of the cutting stock problem objective function. Several papers related to the steel industry formulate and propose solutions to generalize cutting stock problems. Antonio, et al. (1999), Chiotti and Montagna (1998), Chu and Antonio (1999), and Vasko, et al. (1999) deal with the cutting of steel bars into smaller pieces. Steel bars are produced significantly longer than the customer orders and therefore are required to be cut into smaller pieces to fulfill the customer orders.

Several applications outside of the steel industry bear resemblance to SPOF. Both Adelman and Nemhauser (1999) and Adelman, et al. (1999) report on a problem in fiber optic cable manufacturing which is similar to the SPOF in that there are economies of scale as several cable orders can be made from one mother cable and unused portions of the cable may be returned to the inventory for future use. The problem differs in that smaller cables cannot be joined to fulfill larger customer orders.

In Arbib, et al. (2002), a problem in the production of automobile gear belts is presented where a rectangular piece of material is sewn into a sleeve and the sleeve is cut into individual belts. Narrower belts can be sewn together to make wider belts. This is similar to the ability to produce many plates from one slab in SPOF. The problem differs in that there are a limited number of cuts available on a sleeve and a limited number of smaller belts that can be sewn together to make the ordered belts. In SPOF, slabs can be cut an unlimited number of times and the number of plates that can be rolled from one slab to fulfill and order is only limited by the number of plates ordered.

3. INTEGER PROGRAMMING FORMULATION

The formulation of the order fulfillment problem is based on selecting and combining partial burn patterns to complete the customer orders. Each partial burn pattern consists a set of subitems for one order. A subitem is defined as the weight (mass) required to produce a number of plates from a slab. (Note that while slabs are "burned", the length dimension translates directly to a weight dimension. Thus, we deal with tonnage as opposed to size for our discussion.) For example, for a particular order, 26,000 lbs is needed to roll 3 plates at one time, 17,000 lbs is needed to roll 2 plates, and 8,600 lbs is needed to roll 1 plate. Subitems are generated for each order when the order is accepted. The available subitems are constrainted by facility limitations and order size. Each subitem in a burn pattern represents one slab to be rolled on the plate mill.

As an example of a partial burn pattern, consider an order for eight ordered plates with three available subitems, including a 1-plate, 2-plate and 3-plate subitems. A possible partial burn pattern is to burn two 3-plate subitems and one 2-plate subitem from a slab. This partial burn pattern would complete the order by rolling three slabs on the plate mill. The next section discusses the enumeration of partial burn patterns and the following section presents the problem formulation. The goal of the problem formulation is to combine partial burn patterns for different orders into complete burn patterns for the inventory slabs at the lowest cost.

3.1. Partial Burn Pattern Enumeration

It is possible to efficiently enumerate partial burn patterns for feasible order and slab pairs using the implicit enumeration algorithm in Figure 2. The algorithm enumerates the branches of a tree representing all partial burn patterns for the order and slab pair. The algorithm is made efficient

by fathoming branches based on the number of ordered plates and the weight of the slab. The number of plates produced from a partial burn pattern cannot exceed the number of plates required for the order. Also, the required weight of a partial burn pattern cannot exceed the slab weight. The following three partial burn patterns are equivalent, $\{4,4,3\}$, $\{4,3,4\}$, and $\{3,4,4\}$. The algorithm only generates partial burn patterns where the remaining slots are filled with subitems that are the same size or "smaller" than the subitem in the current slot. This significantly reduces the number of partial burn patterns considered and significantly increases the efficiency of the algorithm.

```
sort the subitems from heaviest to lightest
let m be the number of subitems
let p[i] be the plates produced from subitem i
let w[i] be the required weight for subitem i
let s[i] be the subitem in the ith slot
let u be the used weight
let t be the number plates applied
let SW be the slab weight
let OP be the number of ordered plates
let j be the current slot
s[j] = 0  for all slots
j := 1
repeat
    if a subitem is in the current slot ( s[j] > 0 ) then
        u := u - w[s[j]]
        t := t - p[s[j]]
    end if
    s[j] = s[j] + 1
    if s[j] ≤ m then
        u := u + w[s[j]]
        t := t + p[s[j]]
        if u ≤ SW  and t ≤ OP then
            j := j + 1
            s[j] = s[j - 1] - 1
            if s[j] > 0 then
                u := u + w[s[j]]
                t := t + p[s[j]]
            endif
        endif
    else
        s[j] = 0
        save partial burn pattern
    endif
until j := 0
```

Figure 2 – Implicit Enumeration Algorithm Psuedocode

There is a natural dominance between partial burn patterns that fulfill the same number of ordered plates and the dominated partial burn patterns will never appear in an optimal solution of SPOF. Consider the following two partial burn patterns, both producing 10 plates. Pattern 1

burns a slab into 10 pieces with each piece rolled into one ordered plate. Pattern 2 burns a slab into 3 pieces with one piece rolled into 4 order plates and two pieces rolled into 3 order plates. Pattern 1 costs $13,332 and pattern 2 costs $10,183. Pattern 2 is cheaper because of the economies of scale. If the solution calls for ten plates to be supplied from this slab, only the lowest cost partial burn pattern is considered. Therefore, the other ten-plate partial burn patterns are dominated by the lowest cost ten-plate pattern and are removed from the partial burn pattern list.

In the worst case, the above algorithm executes in $O\left(s \cdot (i+1)^{OP}\right)$ time with i being the number of subitems, OP being the number of ordered plates, and s being the number of feasible slabs. Plates are only produced in integer quantities. Therefore, the number of orders plates (OP) and the number of plates produced from each subitem can only be integers. The smallest number of plates that can be produced from a subitem is one. Hence, the maximum number of slots for a partial burn pattern is OP. Each slot can be filled with a subitem or nothing. Therefore, there are i+1 choices for each slot and the worst-case complexity is $O\left((i+1)^{OP}\right)$ or just $O\left(i^{OP}\right)$ for a single order and feasible slab pair. Therefore, for all feasible slabs, the complexity is $O\left(s \cdot i^{OP}\right)$. In practice, the algorithm executes efficiently.

3.2. Pattern-Based Problem Formulation

Variables for the formulation are presented in Table 2, parameters in

Table 3, and objective function in (1) and constraints in (4) through (10).

Table 2 – 0-1 Integer Partial Burn Pattern Based Formulation Variables

Variable	Description
$z_{o,s,p}$	0-1 integer variable representing whether to use partial burn pattern p from slab s for order o.
h_o	0-1 variable representing whether to fulfill order o from new production
u_s	0-1 variable representing whether the remaining weight of slab s is scrap or returns to inventory. 0 – remaining weight can be returned to inventory, 1 – remaining weight is scrap
s_s	Variable representing scrap weight from slab s.
r_s	Variable representing weight from slab s being returned to the inventory.

Table 3 – 0-1 Integer Partial Burn Pattern Based Formulation Constants

Constant	Description
O	Set of orders
S	Set of slabs
$P_{o,s}$	Set of partial burn patterns for order o and slab s
OP_o	Number of ordered plates for order o
$PP_{o,s,p}$	Number of ordered plates produced from partial burn pattern p
$PC_{o,s,p}$	Cost of using partial burn pattern p
$PW_{o,s,p}$	Slab weight required for partial burn pattern p
HC_o	Cost of fulfilling order o from new production
SM_s	Minimum weight of slab s that can be returned to the inventory
SW_s	Slab weight of slab s
SC_s	Cost per pound of steel in slab s
IC	Cost of holding the entire inventory for the complete time horizon

$$\text{Min} \quad \sum_{o\in O}\sum_{s\in S}\sum_{p\in P_{o,s}} PC_{o,s,p}\cdot z_{o,s,p} + \sum_{o\in O} HC_o \cdot h_o + \sum_{s\in S} SC_s \cdot s_s + IC \qquad (1)$$

$$\text{s.t.} \quad \sum_{s\in S}\sum_{p\in P_{o,s}} PP_{o,s,p}\cdot z_{o,s,p} + OP_o \cdot h_o = OP_o \qquad \forall o \in O \qquad (2)$$

$$\sum_{p\in P_{o,s}} z_{o,s,p} \le 1 \qquad \forall (o,s) \in O\times S \qquad (3)$$

$$\sum_{o\in O}\sum_{p\in P_{o,s}} PW_{o,s,p}\cdot z_{o,s,p} + r_s + s_s = SW_s \qquad \forall s \in S \qquad (4)$$

$$r_s + SW_s \cdot u_s \le SW_s \qquad \forall s \in S \qquad (5)$$

$$r_s + SM_s \cdot u_s \geq SM_s \qquad \forall s \in S \qquad (6)$$

$$z_{o,s,p} \in \{0,1\}, \text{ integer} \qquad \forall (o,s) \in O \times S, \ \forall p \in P_{o,s} \qquad (7)$$

$$h_o \in \{0,1\} \qquad \forall o \in O \qquad (8)$$

$$u_s \in \{0,1\} \qquad \forall s \in S \qquad (9)$$

$$s_s \geq 0 \qquad \forall s \in S \qquad (10)$$

The objective function (Equation (1)) minimizes four costs. These costs include, in order, (a) assigning a partial burn pattern to an order, including slab costs, burning costs, and rolling costs; (b) assigning new production to an order, including a service cost due to likely delay. This is achieved with "hypothetical" inventory slabs for assignment; (c) scrap costs; and (d) a constant representing the cost to hold the entire inventory for the entire problem horizon. This inventory holding cost is added because the pattern cost, $PC_{o,s,p}$, includes a reduction for using a slab (or portion thereof) prior to the end of the problem horizon. Consider a fulfillment problem with a horizon of 26 weeks. The cost for holding a slab for the 26 weeks is included in IC. If the slab is feasible for an order with a due date in the 15th week of the problem, at least one pattern will exist. The cost for this pattern, $PC_{o,s,p}$, is reduced by the cost of holding the slab for the remaining 11 weeks of the problem. The net effect on the objective function is to include the cost of holding the slab for 15 weeks. While IC does not affect the optimization, it assures accurate costing for comparing methods.

The constraints can be broken into three subsets; order and pattern constraints, slab constraints, and variable constraints. Equations (2) and (3) are the order and pattern constraints. Equations (2) force the orders to be fulfilled either from existing inventory or from new production. The constraints in (3) limit the number of burn patterns from a slab for a specific

order to one. To illustrate, consider two partial burn patterns from the same slab. Pattern A produces six ordered plates and pattern B produces five ordered plates. If a combination of pattern A and pattern B is possible, there would exist another pattern producing eleven ordered plates. By construction, the eleven plate pattern would cost the same or less than patterns A and B. If no eleven-plate pattern were possible, then the selecting A and B would also be impossible. These constraints enforce this restriction.

Equations (4) through (6) are the slab related constraints. The first equation balances the slab weight used to the initial slab weight. The variables, r_s and s_s, represent the slab weight returned to the inventory and the amount of scrap respectively. Equations (5) and (6) control the feasible values for r_s using the 0-1 variable u_s. If u_s=0, then remaining weight can be returned to the inventory, as Equations (5) and (6) define r_s to be feasible, and s_s is economically driven to zero. If u_s=1, then the remaining weight cannot be returned to the inventory, Equations (5) and (6) force r_s to zero and s_s is the amount of scrap.

The remaining constraints, equations (7) through (10) are the variable constraints. These constraints identify the variable types and allowable values.

Table 4 – Burn Pattern Counts

Data Set	Order Count	Slab Count	Pattern Count
1	2,197	7,075	815,301
2	2,146	7,080	849,920
3	1,996	6,022	639,897
4	2,228	5,448	847,376
5	2,410	4,764	682,100
6	2,782	5,044	614,664
7	4,128	5,150	1,062,724
8	4,179	6,564	1,108,378
9	3,578	5,607	912,606
10	5,793	5,977	1,612,667
11	6,180	6,554	2,393,153
12	5,690	11,814	6,849,396
13	3,928	10,722	4,138,887
14	1,571	9,433	2,066,048
15	2,011	8,211	969,567
16	2,699	8,077	1,222,825
17	2,790	4,038	561,389

Table 4 presents a measure of the problem size for each data set. The "Pattern Count" column is the number of partial burn patterns generated for each problem set. For each order, the formulation contains one 0-1 integer variable and one constraint. For each slab, one 0-1 integer variable, one linear variable and one constraint exists. For each order and slab pair, one constraint is included in the formulation and for each pattern, one 0-1 integer variable exists. The formulation for data set 1 contains 7,075 linear variables, 824,573 0-1 integer variables, and 17,991,875 constraints. The size of the data sets makes it impractical to solve the entire problem at one time. Sections 4, 5, and 6 discuss solving the single order SPOF, present lower and upper bounds to the SPOF based on the single order problem and improvements to the bounds based on Lagrangian Relaxation and Subgradient Optimization.

4. SINGLE ORDER PROBLEM

The single order fulfillment problem is presented in this section because it can be solved quickly in practice and is the basis of the solution methods presented later in this paper. Also, the single order problem is the basis for the solution approach that was used at Bethlehem Steel. The next two sections discuss the formulation and solution method of the single order fulfillment problem.

4.1. Single Order Problem Formulation

Solving a single order problem consists of selecting the optimal combination of partial burn patterns that fulfill the customer order. The following is the formulation of the problem using the variables and constants presented in Table 5 and Table 6 respectively.

Table 5 – Single Order Formulation Variables

Variable	Description
P	Set of partial burn patterns available to fulfill the order (including the hypothetical patterns)
S	Set of slabs form which all of the patterns in P are burned
z_p	0-1 integer variable representing whether to use pattern p to fulfill the order

Table 6 – Single Order Formulation Constants

Constant	Description
PC_p	Cost of using partial burn pattern p to fulfill the order
PP_p	Number of plates produced from partial burn pattern p
OP	Number of plates required to fulfill the order
a_{ps}	0-1 constant representing whether burn pattern p is burned from slab s

$$\text{Min} \quad \sum_{p \in P} PC_p \cdot z_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} PP_p \cdot z_p = OP \tag{12}$$

$$\sum_{p \in P} a_{ps} \cdot z_p \leq 1, \forall s \in S \tag{13}$$

P is the set of partial burn patterns that can be used to fulfill the order. This set includes the hypothetical pattern and therefore, a feasible solution exists for the problem. The set S is the set of slabs from which the partial burn patterns in P are generated. Equation (11) is the objective function that is to minimize the cost of fulfilling the order. Equation (12) makes certain that the order is fulfilled. It is possible (and probable) that several partial burn patterns in P will come from the same slab. The Constraints (13) enforce that only one partial burn pattern from each slab is used. There will be one constraint for each slab in the set S. Slab weight constraints are not needed for the single order problem, as all partial burn patterns generated from the pattern enumeration procedure are feasible.

Table 7 – Single Order Patterns

Pattern	Slab	Plates	Cost
1	Hypothetical	6	$ 13,786
2	803X63890 07V	5	$ 1,693
3	801W00629 09A	5	$ 1,720
4	801W00629 09V	5	$ 1,720
5	803X63890 07V	4	$ 1,462
6	801W00629 09A	4	$ 1,483
7	801W00629 09V	4	$ 1,483
8	803X63890 06V	3	$ 1,219
9	801W00629 09A	3	$ 1,238
10	801W00629 09V	3	$ 1,238
11	803X63890 07V	3	$ 1,520
12	803X63890 06V	2	$ 985
13	801W00629 09A	2	$ 992
14	801W00629 09V	2	$ 992
15	801W00629 09A	1	$ 781
16	801W00629 09V	1	$ 781

A reduction exists for the single order problem based on the quantity of patterns generated for each distinct number of plates. As an example, consider Table 7. In the table, there are three possible patterns for five plates. The order quantity is six plates and, at most, only

one five-plate pattern will be included in any solution. Therefore, not all of the five-plate patterns need to be considered and the lowest cost pattern dominates the remaining patterns. In fact, only

$$\left\lfloor \frac{OP}{PP_p} \right\rfloor$$

patterns need to be included in the solution space. This dominance can be used to reduce the problem and remove patterns 3, 4, 6, 7, 10, and 11.

4.2. Solving the Single Order Problem

An explicit enumeration algorithm based on traversing a tree of all the solutions is used to solve the single order problem. This algorithm generates solutions in the form of $(p_1, p_2, \ldots, p_n)$ where p_i represents a partial burn pattern and the number plates from the patterns in the n-tuple are non-increasing. In other words, $PP_{p_1} \geq PP_{p_2} \geq \ldots \geq PP_{p_n}$. Limiting the enumeration to solutions of this structure guarantees an optimal solution all permutations of $(p_1, p_2, \ldots, p_n)$ are equivalent. The algorithm is outlined in Figure 3.

```
sort the patterns by number of plates decreasing and increasing cost
let c be a pointer to the current pattern in the pattern list
let u[i] be a flag represent whether the slab for pattern I has been used
let p[i] be the number of plates produced from pattern i
let OP be the number ordered plates
let SP be the number of plates produced by the current solution
let s be the number of patterns in the current solution
let s[i] be the ith pattern in the current solution
let n be the number of patterns
c := 1
repeat
    // section 1 – generate the next solution
    repeat
        if u[c] is unused and SP + p[c] ≤ OP
            s := s + 1
            s[s] = c
            mark all other same-slab patterns as used
            SP := SP + p[c]
        end if
        c := c + 1
    until SP = OP or c > n
    // section 2 – possibly save the solution
    if current solution is better than the best then
        save current solution as best
    end if
    // section 3 – backtrack
    repeat
        c := s[s]
        s := s – 1
        SP := SP – p[c]
        mark all other same-slab patterns as unused
        l := p[c]
        repeat
            c := c + 1
        until p[c] ≠ l or c > n
        check for fathomed branch
    until c = 0 or (c < n and branch is not fathomed)
until c = 0
```

Figure 3 - Psuedocode for Solution Enumeration.

The first section of the algorithm generates the next solution. This section runs through the pattern list and adds available patterns to the current solution. The second section checks to see if the current solution is better than the best solution and if true, saves the current solution as the best. The final section executes when a branch is fathomed. The algorithm fathoms branches in three ways: (1) there are no more patterns to add to the solution; (2) the number of plates available in the unused patterns is less than required to complete the order; and (3) a heuristically generated lower bound is larger than the best solution. Tables for the last two methods are created prior to the enumeration.

The overall complexity of the algorithm includes building the tables for the fathoming processes and the algorithm's execution. The table for the plate-based fathoming can be constructed in $O(p)$ where p is the number of patterns. The construction of the lower bound cost table requires a sort for each pattern and is constructed in $O(p^2 \log p)$. Constructing a solution in section one of the algorithm consists of traversing the pattern list once to select the patterns and once to update the used flag for each pattern added to the solution. A solution will have no more than OP patterns because the smallest pattern is for one plate. Therefore, the used flag will only have to be updated OP times. Hence, a solution can be constructed in $O(p + OP \cdot p)$ or $O(OP \cdot p)$. Saving the current solution as the best solution requires copying the current solution that will have no more than OP patterns (because the smallest a pattern can be is for one plate). Therefore, the copy can be completed in $O(OP)$. The final section of the algorithm, the backtrack section, requires a traverse of the pattern list for each pattern removed from the solution. Again, at most, there are OP patterns in the solution and the section can be completed in $O(OP \cdot p)$. The three internal sections of the algorithm are completed in $O(OP \cdot p)$ time. There are at most OP patterns in the solution. There are p possible choices for the first pattern,

p-1 for the second, p-2 for the third, etc. Therefore there are a possible of $p!/OP!$ solutions for the single order problem. If p is larger than OP, then there are on the order of p^{OP} solutions and the outer loop of the algorithm can be executed p^{OP} times. Therefore the overall complexity of the algorithm is $O\left(p^{OP} \cdot OP \cdot p\right)$ or $O\left(p^{OP}\right)$. The construction of the fathoming tables is dominated by the algorithm complexity. In practice, this algorithm executes quickly because of the small size of OP and p and the fathoming on the branches.

5. BOUNDS

As previously stated, the single order problem is the basis for the two bounding methods presented in this section. Both methods solve the single order problem for each order in the data set. The lower bound method relaxes the slab weight constraint allowing overuse of slabs. The upper bound method solves the orders in a priority sequence using a greedy method. Both bounds are discussed below.

5.1. Greedy Lower Bound

The greedy lower bound algorithm, referred to as GreedyLB, is based upon the partial pattern enumeration 0-1 integer programming formulation. The optimal solution to the single order problem is found for each order using the entire inventory. In other words, there is no constraint on overusing a slab. The basic outline of the algorithm is given in Figure 4.

```
repeat for each order
    build a list of partial burn patterns
    reduction of the partial burn patterns
    solve single order problem
until all orders processed
```

Figure 4 - Outline of Greedy Lower Bound procedure.

The first step within the loop is to build the list of partial burn patterns for the current order and the feasible slab ignoring any previous slab usage (Section 3.1). The next step is to reduce the number of burn patterns for the single order problem (Section 4.1). The final step is to solve the single order problem (Section 4.2). This process is repeated for each order.

The complexity of this algorithm is as follows. Building the partial burn patterns for an order has the complexity of $O(s \cdot i^{OP})$ where s is the number of slabs, i is the average number of subitems per order, and OP is the average number of ordered plates. See section 3.1 for a discussion of the complexity of enumerating partial burn patterns. Removing dominated patterns requires a sort and therefore has a complexity of $O(p \log p)$. Finally, solving the single order problem has a complexity of $O(p^{OP})$ (see section 4.2). Each of the loop steps is executed o times. The overall complexity is stated as $O(o \cdot s \cdot i^{OP} + o \cdot p \log p + o \cdot p^{OP})$, or $O(o \cdot (s \cdot i^{OP} + p \log p + p^{OP}))$. In practice, the entire algorithm executes quickly.

5.2. Greedy Upper Bound

The upper bound algorithm, referred to as GreedyUB, is similar to the GreedyLB algorithm except that the slab usage is updated after each order is processed. This algorithm also sequences the orders by priority and the probability of completion from the inventory and processes the orders in this sequence. The basic outline of the algorithm is given in Figure 5.

```
build a list of feasible slabs for each order and update completion
    probability
sort the orders by priority (schedule week) and completion probability
repeat for each order
    build a list of partial burn patterns
    reduction of the partial burn patterns
    solve single order problem
    update inventory with slab usage
until all orders processed
```

Figure 5 – Outline of Greedy Upper Bound Procedure

This algorithm is basically a greedy heuristic. The orders are sequenced by priority and processed in that sequence. The priority is a combination of the due date and the completion probability (based on the tons of feasible slabs in the inventory). Slabs are applied to the orders using the single order problem. The first order has the entire inventory as possible applications. The second order uses the remaining inventory after the applications to the first order. The third order uses the remaining inventory after the applications to the first and second order, etc. As such, this algorithm produces an upper bound to the order fulfillment problem.

The complexity of this algorithm is built up from the individual steps. Building the lists of feasible slabs and updating the completion probability has a complexity of $O(s \cdot o)$ where s is the number of slabs and o is the number of orders. The sort of the orders has a complexity of $O(o \log o)$. Enumerating the partial burn patterns for an order has the complexity of $O(s \cdot i^{OP})$ where i is the average number of subitems per order and OP is the average number of ordered plates. See section 4.1 for a discussion of the complexity of enumerated partial burn patterns. Removing dominated patterns requires a sort and therefore has a complexity of $O(p \log p)$. Finally, solving the single order problem has a complexity of $O(p^{OP})$ (see section 4.2). Each of the loop steps is executed o times. The overall complexity is stated as $O(s \cdot o + o \log o + o \cdot s \cdot i^{OP} + o \cdot p \log p + o \cdot p^{OP})$, or $O(o \cdot (s + \log o + s \cdot i^{OP} + p \log p + p^{OP}))$.

As with GreedyLB, the algorithm executes quickly in practice. This is critical as it is similar to the solution procedure that was used in day-to-day operations at Bethlehem Steel.

6. LAGRANGIAN RELAXATION

This section describes the application of Lagrangian relaxation and Subgradient Optimization to improve the lower and upper bounds.

6.1. Subgradient Optimization Lower Bound

Lagrangian Relaxation and Subgradient Optimization are used to generate improved lower bounds for SPOF. The constraints represented by equation (4) are added to the objective function using Lagrangian Relaxation. Since there is a constraint for each slab $s \in S$, there is a multiplier for each $s \in S$. Equations (14) to (21) present the Lagrangian Relaxation formulation.

$$\text{Min} \quad \sum_{o\in O}\sum_{s\in S}\sum_{p\in P_{o,s}}\left(PC_{o,s,p} - \left(SC_s - \lambda_s\right)\cdot PW_{o,s,p}\right)\cdot z_{o,s,p} + \sum_{s\in S}\left(SC_s - \lambda_s\right)\cdot SW_s - \sum_{s\in S}\left(SC_s - \lambda_s\right)\cdot r_s + IC \tag{14}$$

$$\text{s.t.} \quad \sum_{s\in S}\sum_{p\in P_{o,s}} PP_{o,s,p}\cdot z_{o,s,p} + OP_o \cdot h_o = OP_o \qquad \forall o \in O \tag{15}$$

$$\sum_{p\in P_{o,s}} z_{o,s,p} \le 1 \qquad \forall (o,s) \in O\times S \tag{16}$$

$$r_s + SW_s \cdot u_s \le SW_s \qquad \forall s \in S \tag{17}$$

$$r_s + SM_s \cdot u_s \ge SM_s \qquad \forall s \in S \tag{18}$$

$$z_{o,s,p} \in \{0,1\}, \text{ integer} \qquad \forall (o,s) \in O\times S,\ \forall p \in P_{o,s} \tag{19}$$

$$h_o \in \{0,1\} \qquad \forall o \in O \tag{20}$$

$$u_s \in \{0,1\} \qquad \forall s \in S \tag{21}$$

The constraints represented by equation (4) tied together the original formulation. With these constraints removed, the Lagrangian Relaxation formulation is separable by order. The objective function also includes a constant expression, $\sum_{s\in S}\left(SC_s - \lambda_s\right)\cdot SW_s - IC$.

For each order $o \in O$, the following is the formulation.

$$\text{Min} \quad \sum_{s\in S}\sum_{p\in P_{o,s}}\left(PC_{o,s,p} - \left(SC_s - \lambda_s\right)\cdot PW_{o,s,p}\right)\cdot z_{o,s,p} \tag{22}$$

$$\text{s.t.} \quad \sum_{s\in S}\sum_{p\in P_{o,s}} PP_{o,s,p}\cdot z_{o,s,p} + OP_o \cdot h_o = OP_o \tag{23}$$

$$\sum_{p\in P_{o,s}} z_{o,s,p} \le 1 \qquad \forall s \in S \tag{24}$$

$$z_{o,s,p} \in \{0,1\}, \text{ integer} \qquad \forall s \in S,\ \forall p \in P_{o,s} \tag{25}$$

$$h_o \in \{0,1\} \qquad \forall o \in O \tag{26}$$

The remaining weight formulation is as follows.

$$\text{Min} \quad -\sum_{s\in S}\left(SC_s - \lambda_s\right)\cdot r_s \tag{27}$$

$$\text{s.t.} \quad r_s + SW_s \cdot u_s \le SW_s \qquad \forall s \in S \tag{28}$$

$$r_s + SM_s \cdot u_s \ge SM_s \qquad \forall s \in S \tag{29}$$

$$u_s \in \{0,1\} \qquad \forall s \in S \tag{30}$$

This problem can be solved by inspection. If $SC_s - \lambda_s > 0$, then r_s will attain its maximum value, $r_s = SW_s$ and $u_s = 0$. If $SC_s - \lambda_s \le 0$, then r_s will attain its minimum value, $r_s = 0$ and $u_s = 1$.

This method follows the procedure outlined in Beasley (1993). The termination parameter π is initialized to 2 and is halved after 10 subgradient iterations with no lower bound improvement (the parameter N is set to 10). The procedure is terminated when $\pi < .005$. The method is implemented with the relaxed constraints not being scaled (Subgrad) and scaled (SubgradS).

6.2. Subgradient Optimization Based Upper Bound

During the programming of the greedy upper bound method, it was found that the sequence (sort order) of the patterns is extremely important. For example, consider two partial burn patterns with the same cost and producing the same number of ordered plates. The only difference between the patterns is that they are burned from different slabs. When solving a single order problem, both of these burn patterns are equivalent. When solving the entire problem using the GreedyUB, the decision to use or not use a pattern affects future slab availability. Using the first pattern may produce a higher cost solution than using the second pattern. The reason that the solutions are different is because the patterns appear in a different sort order. The question becomes, can better solutions be found by changing the sort order of the partial burn patterns?

The λ's from Subgradient Optimization can be used to adjust the cost and the sequence of the burn patterns. The costs are adjusted using the burn pattern coefficient from the Lagrangian Relaxation, $PC_{o,s,p} - (SC_s - \lambda_s) \cdot PW_{o,s,p}$. The GreedyUB method is executed and a solution is constructed using the adjusted pattern costs. The cost of the GreedyUB solution is then generated using the original (unadjusted) pattern costs. The λ's change the sort order of the patterns in the single order problems. Therefore, the solution generated from the greedy upper bound method is different for different λ's.

The Subgradient Optimization based upper bound method combines Subgradient Optimization and GreedyUB. Each time an improved lower bound is found during the Subgradient Optimization procedure, a new upper bound is calculated. First, the λ's are used to adjust the costs of the partial burn patterns as described above. Next, GreedyUB is executed using the adjusted costs and a new upper bound is generated. The solution is evaluated using the original pattern costs and the best upper bound is updated if an improved bound is found. Since

the upper bound changes, this algorithm could also find improved lower bounds. Two versions of this procedure are also used; (1) SubgradUB which does not scale the relaxed constraints and (2) SubgradUBS which does scale the relaxed constraints.

7. COMPUATATIONAL RESULTS

7.1. Data Sets

Seventeen real world data sets, spanning several years, have been collected to test the algorithms. Each data set is a snapshot of the slab inventory and the existing orders. The amount of time between the snapshots ranges from one month to over one year. As such, individual orders and slabs may appear in more than one data set.

Table 8 displays information on each data set. The "Order Count" column is the total number of existing orders. The "Week Range" column is the number of weeks into the future in which orders appear. The "Slab Count" column is the number of slabs in the inventory.

Table 8 – Data Set Counts

Data Set	Week Range	Order Count	Slab Count
1	25	2,197	7,075
2	24	2,146	7,080
3	21	1,996	6,022
4	14	2,228	5,448
5	13	2,410	4,764
6	18	2,782	5,044
7	15	4,128	5,150
8	19	4,179	6,564
9	24	3,578	5,607
10	30	5,793	5,977
11	25	6,180	6,554
12	19	5,690	11,814
13	19	3,928	10,722
14	52	1,571	9,433
15	38	2,011	8,211
16	37	2,699	8,077
17	33	2,790	4,038

The individual data sets can be partitioned into smaller problems. Consider two orders, A and B. If order A and slab 1 are a feasible pair and order B and slab 1 are a feasible pair, then orders A and B have related properties and can be placed into the same partition. If there is no intersection between the set of feasible slabs for order A and the set of feasible slabs for order B, then the orders can be placed into separate partitions. By comparing each order's set of feasible slabs, the orders can be partitioned.

A real-world implementation of an algorithm for the SPOF must complete within several hours, or overnight at worse, in order to be used for day-to-day operations. The expected execution times of the solution methods (especially for the larger data sets, 11, 12 and 13) are unreasonable for a real-world implementation. It is necessary to explore reductions to the problem size and a natural reduction for this problem is to reduce the time horizon. (See Huegler 2003 for a review and selection of possible horizons.) The results presented below are for a limited time horizon of three weeks into the future. Using this limitation coincides well with the

overall production scheduling process. Future orders (especially over three weeks into the future) are rarely used in the day-to-day scheduling activities. Additionally, rolling plates prior to due date can cause additional handling and storage issues. Table 9 contains the limited time horizon information on each data set.

Table 9 – Limited Time Horizon Data Set Counts

Data Set	Week Range	Order Count	Slab Count
1	25	722	7,075
2	24	1,198	7,080
3	21	991	6,022
4	14	693	5,448
5	13	810	4,764
6	18	789	5,044
7	15	1,205	5,150
8	19	1,049	6,564
9	24	1,151	5,607
10	30	963	5,977
11	25	903	6,554
12	19	1,281	11,814
13	19	1,131	10,722
14	52	884	9,433
15	38	642	8,211
16	37	689	8,077
17	33	1,043	4,038

7.2. Problem Costs

Table 1 contains the six areas of costs considered for the SPOF. While actual costs were unavailable for the SPOF, we generated costs to emulate their existing structure. This section discusses the costs used for the results.

The slab cost is the cost of raw materials (steel) to make the mother plate which is dependent on the steel properties. Steel produced to tighter restrictions or with extra properties requires additional processing and therefore, costs more to produce. Slabs are grouped into families (called grades) based on their steel properties. The grade costs are randomly generated based upon the frequency the grades appear in the slab inventory. The frequency of a grade

appearing in the inventory is generally inversely proportional to the production cost. High cost steel is ordered less and therefore produced less than lower cost steel. The costs for the grades appearing in the top third by frequency are uniformly selected from the cost range [$100, $200]. The costs for the grades appearing in the middle third by frequency are uniformly selected from the cost range [$150, $250]. The costs for the remaining grades are uniformly selected from the cost range [$200, $300]. Two modifiers representing tighter property restrictions and additional processing increase the grade costs. The modifier costs are selected from the cost ranges [$15, $50] and cost range [$5, $35] respectively. The slab cost per ton is calculated by adding the grade and modifier costs.

The rolling cost is made up of two components: (1) slab heating costs and (2) the cost to roll the slab on the plate mill. The cost to roll a slab is set at $450 with a $2/ton cost to heat the slab with a $3/ton premium for oversized slabs (they require special handling).

The inventory holding cost represents the opportunity cost of cash tied up in inventory. The cost used here is 11% per year or .212% per week. To calculate the holding cost of a slab for one week, multiply the slab weight by the grade cost by .212%.

The burning cost is the cost associated with burning a slab into two pieces. There are two components to the burning cost, setup cost and yield loss. The burning set up cost is $50 per burn with yield loss measured in weight and the cost calculated using the grade cost.

The customer service cost and the new production costs are fixed costs. The customer service cost is incurred when an order cannot be completely fulfilled on time. The new production cost is incurred when caster production is ordered. The customer service cost and new production cost are $10,000 and $2,000 respectively.

7.3. Implementation

All of the algorithms are coded in C and C++ and compiled using the Microsoft Visual C++ 6.0 compiler. The single order linear programs in the Subgradient Optimization based solution methods are solved using the GNU Linear Programming Kit version 3.2.3. The data was collected on a 900 MHz Pentium 4 PC with 384 megabytes of memory running Windows 2000 Professional.

Each algorithm executes the single order problem many times. Generally the single order problem solution procedure executes efficiently but on occasion takes excessive time. Therefore, the execution time is limited to 3 minutes for an individual problem. Also, the Subgradient Optimization algorithms are limited to 2 hours of execution time per partition.

7.4. Results

Table 10 presents the gaps between the GreedyLB lower bound and the GreedyUB upper bound for each of the 17 data sets. The gap in this table is used to calculate the percent reductions in Table 11 through Table 14.

Table 10 – Initial Gap

Data Set	GreedyLB	GreedyUB	Gap
1	$ 7,848,801	$ 8,019,435	$ 170,634
2	$ 12,155,865	$ 12,638,926	$ 483,061
3	$ 11,223,497	$ 11,876,023	$ 652,526
4	$ 7,104,387	$ 7,643,929	$ 539,542
5	$ 9,478,161	$ 9,819,914	$ 341,753
6	$ 8,817,151	$ 9,479,931	$ 662,780
7	$ 14,035,253	$ 15,738,801	$ 1,703,548
8	$ 12,757,340	$ 13,636,104	$ 878,764
9	$ 13,456,737	$ 14,276,855	$ 820,118
10	$ 11,139,463	$ 12,013,858	$ 874,395
11	$ 11,217,555	$ 12,241,776	$ 1,024,221
12	$ 17,038,781	$ 17,643,655	$ 604,874
13	$ 12,163,659	$ 12,677,546	$ 513,887
14	$ 9,475,945	$ 9,722,681	$ 246,736
15	$ 5,214,654	$ 5,498,562	$ 283,908
16	$ 6,558,847	$ 6,909,098	$ 350,251
17	$ 10,541,097	$ 11,658,410	$ 1,117,313
Overall	$ 180,227,193	$ 191,495,504	$ 11,268,311

Table 11 displays the gap reduction for the Subgradient Optimization procedure without scaling (Subgrad). In general, the gap is improved just over 30%. The "Reduction" column contains the percent reduction and is the difference between the gap in this table and the gap in Table 10 divided by the gap in Table 10.

Table 11 – SubgradUB Gap Improvement

Data Set	Subgrad	GreedyUB	Gap	Reduction
1	$ 7,899,878	$ 8,019,435	$ 119,557	29.93%
2	$ 12,235,387	$ 12,638,926	$ 403,539	16.46%
3	$ 11,382,946	$ 11,876,023	$ 493,077	24.44%
4	$ 7,250,616	$ 7,643,929	$ 393,313	27.10%
5	$ 9,562,936	$ 9,819,914	$ 256,978	24.81%
6	$ 9,056,792	$ 9,479,931	$ 423,139	36.16%
7	$ 14,704,604	$ 15,738,801	$ 1,034,197	39.29%
8	$ 13,047,236	$ 13,636,104	$ 588,868	32.99%
9	$ 13,779,871	$ 14,276,855	$ 496,984	39.40%
10	$ 11,501,654	$ 12,013,858	$ 512,204	41.42%
11	$ 11,553,417	$ 12,241,776	$ 688,359	32.79%
12	$ 17,213,287	$ 17,643,655	$ 430,368	28.85%
13	$ 12,201,493	$ 12,677,546	$ 476,053	7.36%
14	$ 9,497,674	$ 9,722,681	$ 225,007	8.81%
15	$ 5,243,299	$ 5,498,562	$ 255,263	10.09%
16	$ 6,672,983	$ 6,909,098	$ 236,115	32.59%
17	$ 10,898,841	$ 11,658,410	$ 759,569	32.02%
Overall	$ 183,702,914	$ 191,495,504	$ 7,792,590	30.85%

Table 12 presents the gap improvement from the same algorithm with scaling (SubgradS). Using scaling reduces the gap about another 1%.

Table 12 – SubgradUBS Gap Improvement

Data Set	SubgradS	GreedyUB	Gap	Reduction
1	$ 7,900,118	$ 8,019,435	$ 119,317	30.07%
2	$ 12,243,086	$ 12,638,926	$ 395,840	18.06%
3	$ 11,382,958	$ 11,876,023	$ 493,065	24.44%
4	$ 7,250,794	$ 7,643,929	$ 393,135	27.14%
5	$ 9,563,727	$ 9,819,914	$ 256,187	25.04%
6	$ 9,056,956	$ 9,479,931	$ 422,975	36.18%
7	$ 14,701,867	$ 15,738,801	$ 1,036,934	39.13%
8	$ 13,053,544	$ 13,636,104	$ 582,560	33.71%
9	$ 13,785,545	$ 14,276,855	$ 491,310	40.09%
10	$ 11,491,324	$ 12,013,858	$ 522,534	40.24%
11	$ 11,557,429	$ 12,241,776	$ 684,347	33.18%
12	$ 17,217,627	$ 17,643,655	$ 426,028	29.57%
13	$ 12,211,155	$ 12,677,546	$ 466,391	9.24%
14	$ 9,498,561	$ 9,722,681	$ 224,120	9.17%
15	$ 5,253,185	$ 5,498,562	$ 245,377	13.57%
16	$ 6,673,331	$ 6,909,098	$ 235,767	32.69%
17	$ 10,901,940	$ 11,658,410	$ 756,470	32.30%
Overall	$ 183,743,147	$ 191,495,504	$ 7,752,357	31.20%

Table 13 shows the improvement from the SubgradUB algorithm without scaling. This method is comparable to Subgrad with an improvement of just over 30%.

Table 13 – SubgradUB Gap Improvement

Data Set	SubgradUB	GreedyUB	Gap	Reduction
1	$ 7,900,164	$ 8,019,435	$ 119,271	30.10%
2	$ 12,235,849	$ 12,638,926	$ 403,077	16.56%
3	$ 11,388,169	$ 11,876,023	$ 487,854	25.24%
4	$ 7,251,302	$ 7,643,929	$ 392,627	27.23%
5	$ 9,563,831	$ 9,819,914	$ 256,083	25.07%
6	$ 9,057,905	$ 9,479,931	$ 422,026	36.32%
7	$ 14,708,135	$ 15,738,801	$ 1,030,666	39.50%
8	$ 13,042,829	$ 13,636,104	$ 593,275	32.49%
9	$ 13,782,592	$ 14,276,855	$ 494,263	39.73%
10	$ 11,501,932	$ 12,013,858	$ 511,926	41.45%
11	$ 11,555,267	$ 12,241,776	$ 686,509	32.97%
12	$ 17,214,463	$ 17,643,655	$ 429,192	29.04%
13	$ 12,201,720	$ 12,677,546	$ 475,826	7.41%
14	$ 9,498,465	$ 9,722,681	$ 224,216	9.13%
15	$ 5,248,101	$ 5,498,562	$ 250,461	11.78%
16	$ 6,634,637	$ 6,909,098	$ 274,461	21.64%
17	$ 10,905,197	$ 11,658,410	$ 753,213	32.59%
Overall	$ 183,690,558	$ 191,495,504	$ 7,804,946	30.74%

Table 14 presents the results from the SubgradUB with scaling, or SubgradUBS. This performs slightly better than SubgradS, with a gap reduction of 31.33%.

Table 14 – SubgradUBS Gap Improvement

Data Set	SubgradUBS	GreedyUB	Gap	Reduction
1	$ 7,900,214	$ 8,019,435	$ 119,221	30.13%
2	$ 12,243,098	$ 12,638,926	$ 395,828	18.06%
3	$ 11,386,060	$ 11,876,023	$ 489,963	24.91%
4	$ 7,251,784	$ 7,643,929	$ 392,145	27.32%
5	$ 9,564,153	$ 9,819,914	$ 255,761	25.16%
6	$ 9,058,701	$ 9,479,931	$ 421,230	36.44%
7	$ 14,706,451	$ 15,738,801	$ 1,032,350	39.40%
8	$ 13,054,073	$ 13,636,104	$ 582,031	33.77%
9	$ 13,787,461	$ 14,276,855	$ 489,394	40.33%
10	$ 11,492,536	$ 12,013,858	$ 521,322	40.38%
11	$ 11,559,557	$ 12,241,776	$ 682,219	33.39%
12	$ 17,214,791	$ 17,643,655	$ 428,864	29.10%
13	$ 12,211,164	$ 12,677,546	$ 466,382	9.24%
14	$ 9,499,097	$ 9,722,681	$ 223,584	9.38%
15	$ 5,255,557	$ 5,498,562	$ 243,005	14.41%
16	$ 6,667,754	$ 6,909,098	$ 241,344	31.09%
17	$ 10,904,639	$ 11,658,410	$ 753,771	32.54%
Overall	$ 183,757,090	$ 191,495,504	$ 7,738,414	31.33%

Table 15 presents the gap between the best lower bound and the GreedyUB upper bound. The best lower bound, BestLB, is calculated from Subgrad, SubgradS, SubgradUB, and SubgradUBS. The gap in this table is used to calculate the percent reduction in Table 16 and Table 17.

Table 15 – BestLB Gap

Data Set	BestLB	GreedyUB	Gap
1	$ 7,900,447	$ 8,019,435	$ 118,988
2	$ 12,243,567	$ 12,638,926	$ 395,359
3	$ 11,388,380	$ 11,876,023	$ 487,643
4	$ 7,251,908	$ 7,643,929	$ 392,021
5	$ 9,564,308	$ 9,819,914	$ 255,606
6	$ 9,058,992	$ 9,479,931	$ 420,939
7	$ 14,709,134	$ 15,738,801	$ 1,029,667
8	$ 13,054,527	$ 13,636,104	$ 581,577
9	$ 13,787,678	$ 14,276,855	$ 489,177
10	$ 11,503,327	$ 12,013,858	$ 510,531
11	$ 11,559,759	$ 12,241,776	$ 682,017
12	$ 17,218,573	$ 17,643,655	$ 425,082
13	$ 12,211,785	$ 12,677,546	$ 465,761
14	$ 9,499,527	$ 9,722,681	$ 223,154
15	$ 5,255,586	$ 5,498,562	$ 242,976
16	$ 6,673,999	$ 6,909,098	$ 235,099
17	$ 10,905,504	$ 11,658,410	$ 752,906
Overall	$ 183,787,001	$ 191,495,504	$ 7,708,503

Table 16 contains the upper bound improvements from the SubgradUB without scaling. Overall, the gap is improved close to 34%, which represents an actual savings of over the current solution method of about $3.8 million over the three-week period, or an average of 2% for the 17 data sets.

Table 16 – SubgradUB Gap Improvement

Data Set	BestLB	SubgradUB	Gap	Reduction	Savings
1	$ 7,900,447	$ 7,976,994	$ 76,547	24.87%	0.53%
2	$ 12,243,567	$ 12,508,095	$ 264,528	27.08%	1.04%
3	$ 11,388,380	$ 11,666,330	$ 277,950	32.14%	1.77%
4	$ 7,251,908	$ 7,420,661	$ 168,753	41.38%	2.92%
5	$ 9,564,308	$ 9,727,599	$ 163,291	27.01%	0.94%
6	$ 9,058,992	$ 9,246,751	$ 187,759	35.18%	2.46%
7	$ 14,709,134	$ 15,052,807	$ 343,673	40.27%	4.36%
8	$ 13,054,527	$ 13,321,359	$ 266,832	35.82%	2.31%
9	$ 13,787,678	$ 14,024,464	$ 236,786	30.77%	1.77%
10	$ 11,503,327	$ 11,802,062	$ 298,735	24.22%	1.76%
11	$ 11,559,759	$ 11,815,166	$ 255,407	41.65%	3.48%
12	$ 17,218,573	$ 17,453,050	$ 234,477	31.51%	1.08%
13	$ 12,211,785	$ 12,460,359	$ 248,574	42.26%	1.71%
14	$ 9,499,527	$ 9,679,553	$ 180,026	17.48%	0.44%
15	$ 5,255,586	$ 5,466,954	$ 211,368	11.13%	0.57%
16	$ 6,673,999	$ 6,893,330	$ 219,331	4.50%	0.23%
17	$ 10,905,504	$ 11,155,013	$ 249,509	45.05%	4.32%
Overall	$ 183,787,001	$ 187,670,547	$ 3,883,546	33.94%	2.00%

Table 17 presents improvements from the SubgradUB with scaling. Overall, the gap is improved over 35%. This represents an actual savings of close to $4.0 million or 2.07 % for the 17 data sets.

Table 17 – SubgradUBS Gap Improvement

Data Set	BestLB	SubgradUBS	Gap	Reduction	Savings
1	$ 7,900,447	$ 7,976,262	$ 75,815	25.30%	0.54%
2	$ 12,243,567	$ 12,493,374	$ 249,807	30.13%	1.15%
3	$ 11,388,380	$ 11,644,942	$ 256,562	35.41%	1.95%
4	$ 7,251,908	$ 7,413,756	$ 161,848	42.66%	3.01%
5	$ 9,564,308	$ 9,720,565	$ 156,257	29.07%	1.01%
6	$ 9,058,992	$ 9,235,691	$ 176,699	36.85%	2.58%
7	$ 14,709,134	$ 15,053,359	$ 344,225	40.24%	4.36%
8	$ 13,054,527	$ 13,318,331	$ 263,804	36.16%	2.33%
9	$ 13,787,678	$ 14,023,268	$ 235,590	30.92%	1.78%
10	$ 11,503,327	$ 11,800,577	$ 297,250	24.39%	1.78%
11	$ 11,559,759	$ 11,789,884	$ 230,125	44.12%	3.69%
12	$ 17,218,573	$ 17,439,308	$ 220,735	33.78%	1.16%
13	$ 12,211,785	$ 12,458,409	$ 246,624	42.64%	1.73%
14	$ 9,499,527	$ 9,677,403	$ 177,876	18.35%	0.47%
15	$ 5,255,586	$ 5,455,175	$ 199,589	15.28%	0.79%
16	$ 6,673,999	$ 6,881,741	$ 207,742	7.81%	0.40%
17	$ 10,905,504	$ 11,150,225	$ 244,721	45.48%	4.36%
Overall	$ 183,787,001	$ 187,532,270	$ 3,745,269	35.17%	2.07%

Table 18 displays the average execution times for each algorithm. The times are presented in seconds. Using scaling improves the execution times of the Subgradient based methods. However, even with the limited time horizon, the Subgradient based algorithms execute slowly on the larger data sets.

Table 18 – Average Execution Times (HH:MM:SS.SS)

Data Set	GreedyLB	GreedyUB	Subgrad	SubgradS	SubgradUB	SubgradUBS
1	00:00:06.41	00:00:06.37	01:23:27.50	01:11:15.81	02:14:36.78	01:48:32.68
2	00:00:51.23	00:00:49.90	01:57:52.86	02:28:39.89	02:59:26.92	02:31:53.12
3	00:00:08.32	00:00:07.72	00:57:21.65	00:41:43.95	01:25:11.27	01:04:03.80
4	00:00:04.44	00:00:04.81	00:25:36.66	00:24:52.10	00:43:19.95	00:36:51.25
5	00:00:05.44	00:00:05.97	00:28:25.94	00:25:15.29	00:48:45.31	01:31:20.77
6	00:00:08.77	00:00:07.99	00:33:58.90	00:28:50.08	00:53:41.63	01:00:52.17
7	00:00:12.69	00:00:09.70	01:02:06.55	00:58:35.24	01:49:22.81	01:37:18.80
8	00:01:21.54	00:01:17.60	00:56:42.97	00:51:09.25	03:21:13.66	02:42:40.05
9	00:01:00.94	00:00:42.62	00:52:07.21	00:48:19.80	02:31:20.16	02:38:32.98
10	00:00:37.26	00:00:25.58	00:34:07.68	00:33:13.73	01:02:53.19	00:46:03.57
11	00:00:10.44	00:00:08.15	00:43:33.93	00:43:06.84	01:00:46.70	01:01:37.46
12	00:00:37.08	00:00:40.23	06:14:35.79	06:16:41.97	06:25:15.41	06:24:08.83
13	00:00:26.41	00:00:27.39	04:57:56.17	04:45:57.68	05:27:45.00	05:20:21.33
14	00:03:25.05	00:03:28.04	03:24:10.74	03:17:28.13	04:53:03.76	04:15:49.53
15	00:00:09.81	00:00:08.16	01:09:00.97	00:53:12.24	02:16:11.97	01:52:51.65
16	00:00:12.33	00:00:11.40	01:12:22.31	01:02:31.70	02:16:13.22	02:16:23.11
17	00:00:09.13	00:00:06.39	00:39:57.85	00:46:03.44	00:58:27.88	01:15:30.08
Average	00:00:34.55	00:00:32.24	01:37:15.63	01:33:56.30	02:25:09.15	02:16:45.36

8. APPLICATION OF THE MULTIPLIERS

The execution of the SubgradUB solution method generates a multiplier value for each slab. These multipliers are used by the GreedyUB solution to adjust the slab costs and generate an improved solution. Figure 6 is a chart comparing the slab multipliers (lambdas) for partition three of data set one. The x-axis is the partition slabs sorted by increasing lambda values. The y-axis is a running percent of slabs in the solution. The graph is read in the following manner, 14.47% (y-axis) of the slabs in the SubgradUB solution have a lambda value of .0518835 (x-axis) or less. Approximately 80% of the slabs in the SubgradUB solution have a lambda value of .12 or above. Larger multiplier values correspond to the desirability of an inventory slab.

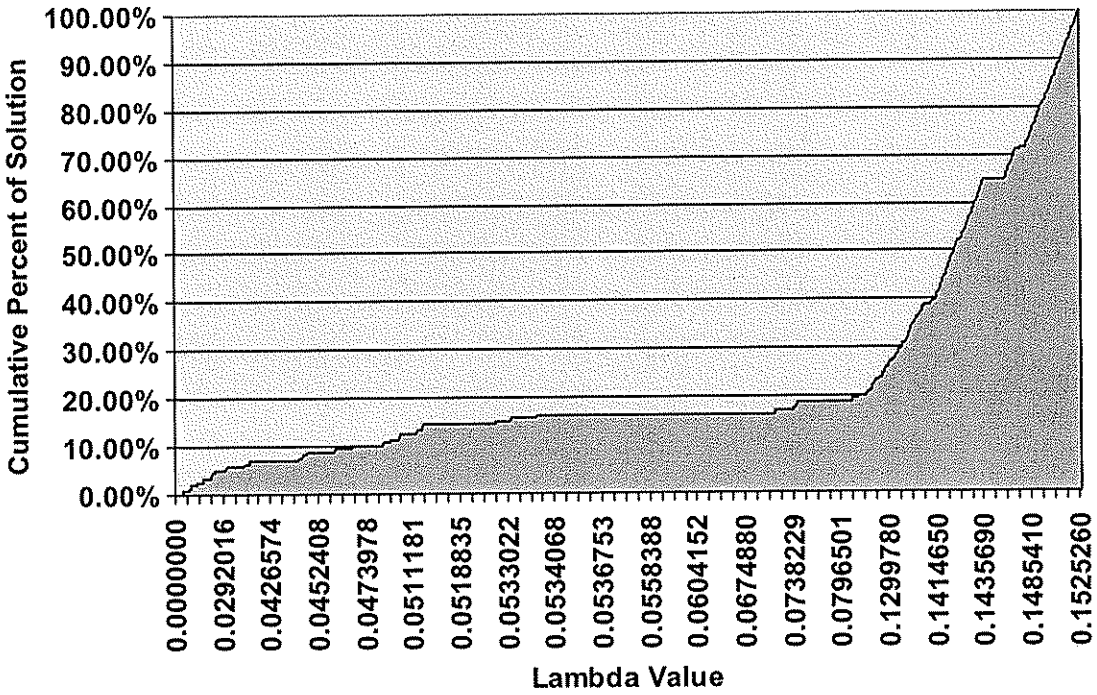

Figure 6 – Cumulative Percent of Solution vs. Multiplier (Lambda) Value

The SubgradUB slab lambdas are useful if a new rush order appears. Rush orders occur regularly and are generally caused by production mistakes. When a rush order appears, instead of running the SubgradUB solution method over, the multipliers from the previous execution can be used by the GreedyUB solution method.

The following procedure is used to test the effects of using the multipliers: A rush order is added to the data set and the GreedyUB algorithm is executed with and without using multipliers. The results are then compared.

The time horizon for the data sets was again restricted to three weeks. The orders with due dates beyond the 3 week time horizon are used as rush orders. One at a time, each order is considered as a rush order. Table 19 and Table 20 present the basic results of using the multipliers. The "Improved" column contains the number of times using the multipliers produced an improved solution over not using the multipliers. The "Total" column is the number

of different rush orders. Overall, using the multipliers improved the rush order solutions around 97% of the time, 96.84% using lambdas from SubgradUB and 97.15% using multipliers from SubgradUBS.

Table 19 – Rush Order Improvement with Multipliers from No Scaling

Data Set	Improved	Total	Percent
1	218	239	91.21%
2	760	786	96.69%
3	1,227	1,263	97.15%
4	1,035	1,105	93.67%
5	969	1,001	96.80%
6	1,039	1,075	96.65%
7	2,436	2,516	96.82%
8	2,382	2,486	95.82%
9	1,698	1,722	98.61%
10	5,367	5,398	99.43%
11	3,146	3,295	95.48%
12	1,819	1,893	96.09%
13	1,029	1,065	96.62%
14	134	147	91.16%
15	226	255	88.63%
16	389	427	91.10%
17	1,081	1,096	98.63%
Overall	24,955	25,769	96.84%

Table 20 – Rush Order Improvement with Multipliers from Scaling

Data Set	Improved	Count	Percent
1	363	429	84.62%
2	778	807	96.41%
3	822	859	95.69%
4	1,096	1,172	93.52%
5	1,143	1,200	95.25%
6	1,397	1,496	93.38%
7	2,352	2,380	98.82%
8	2,452	2,521	97.26%
9	1,692	1,722	98.26%
10	3,213	3,229	99.50%
11	4,089	4,282	95.49%
12	2,587	2,605	99.31%
13	1,033	1,039	99.42%
14	155	167	92.81%
15	1,075	1,091	98.53%
16	2,116	2,136	99.06%
17	1,205	1,241	97.10%
Overall	27,568	28,376	97.15%

Table 21 and Table 22 present the results. The "Count" column it the number of rush orders considered for each data set. The "Improved" column contains the percent of the time using the multipliers improved the GreedyUB solution. The "Savings" column contains the cost savings from using the multipliers. Table 21 presents the results using the multipliers generated from the no scaling version; SubgradUB while Table 22 presents the results using the scaling version, SubgradUBS.

Table 21 – Rush Order Improvements with No Scaling

Data Set	Count	Improved	Savings
1	239	91.21%	0.66%
2	786	96.69%	0.64%
3	1,263	97.15%	1.18%
4	1,105	93.67%	2.28%
5	1,001	96.80%	0.99%
6	1,075	96.65%	2.92%
7	2,516	96.82%	3.60%
8	2,486	95.82%	1.61%
9	1,722	98.61%	1.73%
10	5,398	99.43%	1.18%
11	3,295	95.48%	1.28%
12	1,893	96.09%	1.29%
13	1,065	96.62%	4.97%
14	147	91.16%	0.85%
15	255	88.63%	0.61%
16	427	91.10%	0.54%
17	1,096	98.63%	7.28%
Overall	25,769	96.84%	2.21%

Table 22 – Rush Order Improvements with Scaling

Data Set	Count	Improved	Savings
1	429	84.62%	0.56%
2	807	96.41%	0.85%
3	859	95.69%	1.50%
4	1,172	93.52%	2.48%
5	1,200	95.25%	1.12%
6	1,496	93.38%	3.12%
7	2,380	98.82%	3.63%
8	2,521	97.26%	1.87%
9	1,722	98.26%	1.82%
10	3,229	99.50%	1.49%
11	4,282	95.49%	1.40%
12	2,605	99.31%	1.04%
13	1,039	99.42%	4.80%
14	167	92.81%	1.07%
15	1,091	98.53%	0.47%
16	2,136	99.06%	0.40%
17	1,241	97.10%	7.15%
Overall	28,376	97.15%	2.13%

Using the multipliers maintains the level of savings achieved by the SubgradUB algorithms. The savings here are higher than the savings reported in the previous section

because not all of the partitions are included in these results. Only partitions with generated multipliers are included here. The results illustrate, that when no time is available for re-solution of the complete algorithm (over six hours for the larger data sets), using the multiplier values in GreedyUB leads to improved savings. Recall that this algorithm solved in 32 seconds, on average, for the 17 data sets.

9. CONCLUSIONS

This paper presented a 0-1 integer formulation and solution methods for the steel plate order fulfillment problem (SPOF). The SPOF is a generalization of one-dimensional cutting and packing problems.

Three methods are presented for generating and improving lower bounds for the problem, including a greedy heuristic and two methods based on Langrangean Relaxation and Subgradient Optimization. It is shown that the lower bound methods can reduce the solution gaps in excess of 30%. Two similar methods are also presented for generating and improving upper bounds. It is shown that the lower bound methods can reduce the solution gaps in excess of 34%. The upper bound improvements represent a real-world cost reduction of over 2%, or almost $4 million dollars over a three-week period, for the 17 analyzed data sets.

The presented approaches were able to solve real-problems over a three-week period, representing problems with, on average, 6,916 slabs and 950 orders (for the three week time horizon). The solutions were achieved in less than 7 hours on a 900 MHz desktop computer. Furthermore, multiplier values provided by the Subgradient Optimization methods are shown to be useful for handling rush orders such that the time-consuming improvement procedure does not need to be re-executed.

Our models tied the inventory assignment decisions to actual production through use of "hypothetical slabs" in that if inventory could not satisfy an order, new production was required. Future research may look at examining a closer tie between inventory assignment and production scheduling solutions, as they are highly integrated. Unfortunately, this will lead to even larger problems to solve, but may lead to improved solutions and better customer service.

10. REFERENCES

D. Adelman, G. L. Nemhauser. 1999. Price-directed control of remnant inventory systems. *Operations Research* **47** 889-898.

---------, ----------, M. Padron, R. Stubbs, R. Pandit. 1999. Allocating fibers in cable manufacturing. *Manufacturing and Service Operations Management* **1** 21-35.

J. Antonio, F. Chauvet, C. Chu, J-M. Proth. 1999. The cutting stock problem with mixed objectives: Two heuristics based on dynamic programming. *European Journal of Operational Research* **114** 395-402.

C. Arbib, F. Marinelli, F. Rossi, F. Di Iorio. 2002. Cutting and reuse: An application from automobile component manufacturing. *Operations Research* **50** 923-934.

J.E. Beasley. 1993. Lagrangian Relaxation. C.R. Reeves (Ed.), *Modern Heuristic Methods*. Blackwell Scientific Publications, Oxford, 243-303.

O. J. A. Chiotti, J. M. Montagna. 1998. A global approach for the provision of bar pieces in a metallurgic firm. *Mathematical and Computer Modeling* **28** 73-89.

C. Chu, J. Antonio. 1999. Approximation algorithms to solve real-life multicriteria cutting stock problems. *Operations Research* **47** 495-508.

J. Cohen, H-M. Wallmeier, U. Twisselmann, B. Lantz, D. O'Dell. 1984. Enhancing hot mill slab yield, usage and throughput using expert systems and numerical optimization. ISS Technical Paper.

R. W. Haessler. 1978. A procedure for solving then 1.5-dimensional coil slitting problem. *AIIE Transactions* **10** 70-75.

----------, M. A. Vonderembse. 1979. A procedure for solving the master slab cutting stock problem in the steel industry. *AIIE Transactions* **11** 161-165.

P. A. Huegler. 2003. Fulfilling Customer Orders for Steel Plate from Existing Inventory. Ph.D. Dissertation, Industrial and Systems Engineering, Lehigh University, Bethlehem, Pennsylvania.

J. R. Kalagnanam, M. W. Dawande, M. Trumbo, H. S. Lee. 2000. The surplus inventory matching problem in the process industry. *Operations Research* **48** 505-516.

F. J. Vasko, M. L. Creggar, K. L. Stott, L. R. Woodyatt. 1994. Assigning slabs to orders: An example of appropriate model formulation. *Computers and Industrial Engineering* **26** 797-800.

----------, D. D. Newhart, K. L. Stott, Jr. 1999. A hierarchal approach for one-dimensional cutting stock problems in the steel industry that maximizes yield and minimized overgrading. *European Journal of Operational Research* **114** 72-82.

----------, F. E. Wolf, K. L. Stott, O. Ersham Jr. 1992. Bethlehem steel combines cutting stock and set covering to enhance customer service. *Mathematical and Computer Modeling* **16** 9-17.

M. A. Vonderembse. 1995. Exploring a design decision for a cutting stock problem in the steel industry: all design width are not created equal. *IIE Transactions* **27** 358-367.

----------, R. W. Haessler. 1982a. A mathematical programming approach to schedule master slab casters in the steel industry. *Management Science* **28** 1450-1461.

----------, ---------. 1982b. Scheduling master slab casters. Iron and Steel Engineer. 59, 39-43.

L. Tang, J. Liu, A. Rong, Z. Yang. 2001a. An effective heuristic algorithm to minimize stack shuffles in selecting steel slabs from the slab yard for heating and rolling. *Journal of the Operational Research Society* **52** 1091-1097

---------, ----------, ----------, ----------. 2001b. A review of planning and scheduling systems and methods for integrated steel productions. *European Journal of Operational Research* **133** 1-20.