The material allocation problem in the steel industry

H. Yanagisawa

A major challenge in the initial stage of production planning for the steel industry is the material allocation problem (MAP): finding the best match of orders and materials (steel slabs and coils) with respect to an objective function that takes into account order due dates, preferences for matches, allocated weights, surplus weights of materials, and other factors. The MAP is NP-hard and is difficult to solve optimally. We apply a local search algorithm for the MAP that includes rich moves, such as ejection chain methods. Our algorithm is yielding considerable cost reduction in a real steelworks. In particular, a two-variable integer programming (TVIP) neighborhood search technique contributed to the cost reductions. TVIP defines a neighborhood space for the local search as a two-variable integer programming problem and efficiently finds a solution in the neighborhood. By using TVIP, the number of small batches of surplus material can be successfully reduced.

Introduction

Steelworks usually produce considerable surplus inventory, as when orders are canceled after materials have been produced or when some finished products are below the quality levels required for an order. To make use of surplus inventory, daily production planning is typically divided into two main steps. In the first step, the plan tries to satisfy orders as far as possible with existing materials (such as surplus inventory). In the second step, the plan designs production units for the manufacture of the remaining orders.

The material allocation problem (MAP) is a problem in the first step of the order-fulfillment process. In the MAP, the existing materials may include work-in-progress goods and surplus inventory. The work-in-progress goods are tentatively allocated to some orders, but they can be reallocated to new orders while solving the MAP.

The MAP is a kind of matching problem on a sparse bipartite graph, allocating a given set of materials to a given set of orders. Each order restricts its allocatable materials through the quality and other attributes of the order. In **Figure 1(a)**, the dashed line from the material to an order indicates that the material can be allocated to that order. The material can be cut into any size and each order can be satisfied with multiple sources of material. For example, order A and order B respectively require five tons and four tons, and material batch V has a weight of ten tons, which can be allocated to both orders. When we allocate the material to these orders, the material will be split into three parts: five tons for order A, four tons for order B, and one ton remaining as surplus. [The trimming and yield losses shown in **Figure 1(b)** are discussed below in the section on matches]. In the following argument, we refer to *surplus* as the remainder of the material after allocations.

The MAP has several hard constraints that make it difficult to solve. One of the constraints is a *unit weight constraint:* When we allocate a batch of materials to an order, the materials must be cut into pieces whose sizes are within the range specified by the order. For example, suppose that an order requires 12 tons and that maximum and minimum unit weights specified in the order are five tons and four tons, respectively. (Customers place constraints on orders for reasons such as transport limitations and manufacturing conditions.) When we allocate a 12-ton batch of material to the order, the only solution that meets the constraint conditions is to cut it into three parts of four tons each. Other MAP constraints are described in the next section.

©**Copyright** 2007 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/07/$5.00 © 2007 IBM

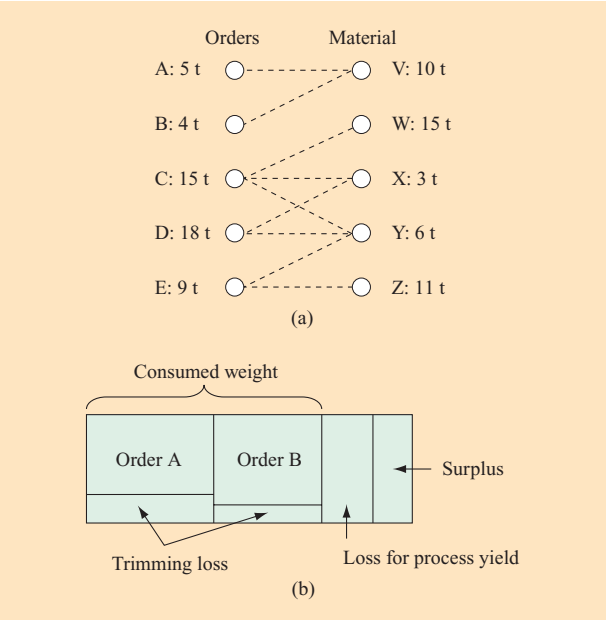

Figure 1

(a) Instance of a MAP (t = tons). (b) Material allocation.

Each solution is evaluated from a broad perspective that includes the due dates of orders, preferences for matches, allocated weights, and the surplus weights of materials. It is evaluated primarily with respect to the total allocated weight, which also leads to reducing the total surplus. Because of the unit weight constraint, many small batches of surplus material remain after these allocations. Though a large surplus offers possibilities for allocation to orders in the near future, a small surplus (typically less than five tons) is hard to allocate because of the unit weight constraint. Thus, a small surplus is likely to be discarded, which is a pure loss for the steel company, so the minimization of the number of small surpluses is an important evaluation item.

It is easy to show that the MAP is NP-hard because the MAP includes a generalized assignment problem [1] as a subproblem. A typical generalized assignment problem is a problem assigning a set of jobs to a set of machines. The jobs and the machines correspond respectively to the orders and the materials in the MAP.

Because the MAP is an NP-hard problem and the sizes of the instances are very large, we use a local search to obtain a good solution. Our algorithm is based on a variable neighborhood search (VNS) [2] strategy. VNS combines local search with systematic changes of neighborhood and escape phases from local optima. The search first explores a small neighborhood around the current solution and allows an exchange with the current solution if a better one is found. This process continues until the search fails to find a better solution. When this happens, the search explores neighborhoods more distant from the current solution and continues the process. For the local search, we use nine types of moves, including ejection chain search [3]. In the local search algorithm, we adopt some acceleration techniques, such as the *don't-look bit* [4], and incorporate simple heuristics for surplus reduction into the local search algorithm.

We found that there are still a number of small surpluses produced by this basic algorithm, but that some of them can be removed by trying extensive searches. Therefore, we use a two-variable integer programming (TVIP) neighborhood search technique to reduce the number of small surpluses. TVIP defines a neighborhood for surplus reduction, and we can show that searching the neighborhoods is essentially equivalent to solving TVIP problems. In addition, we use an efficient algorithm to solve the TVIP problem that combines binary search and an algorithm for lattice-point counting in right triangles. For the lattice-point counting, an algorithm exists [5], but we have developed a simpler one.

The use of our algorithms by a Japanese steel company yielded considerable cost reductions.

The rest of this paper is organized as follows: We describe the details of the MAP, show our basic algorithm for the MAP, and propose a TVIP technique for surplus reduction. We then show our experimental results.

Related problems

The MAP is similar to the generalized assignment problem [1] and the minimum cost flow problem, but these models cannot deal with the unit weight and other constraints.

Kalagnanam et al. [6, 7] modeled the MAP (also known as the *surplus inventory matching problem*) as a multiple knapsack problem with color constraints. Though the problems are similar, our models differ on many points: For example, the objective in the Kalagnanam model is to maximize the total allocated weights, but the objective in our model is to maximize the total profit. However, the crucial difference is that their model cannot deal with the minimization of the number of small surpluses. In [7], Kalagnanam et al. propose a heuristics based on the maximum flow algorithm for surplus reduction, but it is designed to reduce the total surplus, not to reduce the number of small surpluses.

Material allocation problem

In this section, we describe the formulation of the MAP. In the MAP, the set O of orders, the set S of (steel) materials, and the set M of the allocatable matches (pairs of orders and materials) are given. Formally, the MAP is the following optimization problem:

The objective is

$$\max \sum_{i \in O} zo_i + \sum_{j \in S} zs_j - \sum_{(i,j) \in M} zm_{ij},$$

subject to constraints (1) through (8):

$$zo_i = PF_i \min \left\{ TW_i \sum_{(i,j) \in M} aw_{ij} \right\} \quad (\forall i \in O); \tag{1}$$

$$\sum_{(i,j) \in M} aw_{ij} \leq TW_i^{\max} \quad (\forall i \in O); \tag{2}$$

$$n_{ij} PU_i^{\min} \leq aw_{ij} \leq n_{ij} PU_i^{\max} \quad [\forall (i,j) \in M], \tag{3}$$

where

$$n_{ij} \in Z^+ \quad [\forall (i,j) \in M];$$

$$zs_j = PF_j cw_j + C_j f(SW_j - cw_j) \quad (\forall j \in S), \tag{4}$$

where

$$f(x) = c_1 x^{c_2} \exp(-c_3 x^3);$$

$$cw_j \leq SW_j \quad (\forall j \in S); \tag{5}$$

packing constraint; (6)

$$zm_{ij} = C_{ij} aw_{ij} \quad [\forall (i,j) \in M]; \tag{7}$$

and

$$cw_j = SW_j PY_{ij} + \sum_{(i,j) \in M} (aw_{ij}/TL_{ij}) \quad (\forall j \in S), \tag{8}$$

where

$aw_{ij} \geq 0$, $aw_{ij} \in R$ $[\forall (i,j) \in M]$.

The packing constraint (6) is defined in the materials section below.

The variables are zo_i, zs_j, zm_{ij}, aw_{ij}, n_{ij}, and cw_j, and the other terms are labeled constants associated with orders, materials, and matches. The variables have the following meanings:

zo_i : objective value for order i.
zs_j : objective value for material j.
zm_{ij} : objective value for match (i, j).
aw_{ij} : allocated weight of match (i, j).
n_{ij} : number of *units* for match (i, j).
cw_j : allocated weight (including wastes) of material j [for each $i \in O$, $j \in S$, and $(i, j) \in M$].

In the following subsections, we explain constraints (1)–(8).

Orders

Constraints (1)–(3) apply to orders. Each $i \in O$ is associated with the following properties: maximum unit weight $PU_i^{\max}(>0)$, minimum unit weight $PU_i^{\min}(>0)$, target weight TW_i (>0), maximum total allocatable weight $TW_i^{\max}(>0)$, and profit per weight PF_i (>0), where $PU_i^{\min} \leq PU_i^{\max}$ and $TW_i \leq TW_i^{\max}$. The weight TW_i is the required weight for the order i, and $TW_i^{\max}$ is the maximum allocatable weight (so the total allocated weight for the order i cannot exceed $TW_i^{\max}$). This allowance with respect to the target weight is defined to avoid leaving many small surpluses after allocation. The profit per weight PF_i is a parameter for the objective function. An order i that is urgent is associated with a large PF_i.

Equation (1) defines the objective value zo_i for each order i. It is defined by the product of the profit per weight PF_i, the minimum value of the target weight TW_i, and the total allocated weight for the order i,

$$\left[\sum_{(i,j) \in M} aw_{ij} \right].$$

Since the allowance is only for satisfying the unit weight constraint, it is preferable to encourage allocating materials close to the target weight.

Constraint (2) ensures that the total allocated weight

$$\left[\sum_{(i,j) \in M} aw_{ij} \right]$$

does not exceed the maximum allocatable weight $TW_i^{\max}$.

The unit weight constraint (3) restricts the allocated materials so that each batch of material that is allocated to the order i can be cut into pieces in the size range$(PU_i^{\min}, PU_i^{\max})$.

Materials

Constraints (4)–(6) apply to the materials. Each $j \in S$ is associated with the following properties: weight SW_j (>0), profit per weight PF_j (>0), and cost per weight C_j (>0). The SW_j is the weight of the piece of material $j \in S$. The profit per weight PF_j and the cost per weight C_j are parameters for the objective value. A piece of material with a high profit per weight PF_j is favored for allocation. A piece of material with a high cost C_j should not be discarded.

Equation (4) defines the objective value zs_j. The cw_j is the consumed weight of the material j, which can be assumed to be an approximation of the sum of the allocated weights, so

$$cw_j \approx \sum_{(i,j) \in M} aw_{ij}.$$

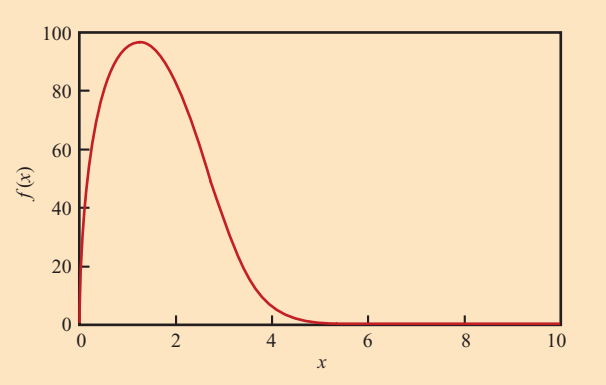

Figure 2

Penalty function $f(x)$ with parameters $c_1 = 100$, $c_2 = 0.3$, and $c_3 = 0.05$.

Table 1 Example of the function g: $M \times M \rightarrow$ {TRUE, FALSE}.

	m_1	m_2	m_3
m_1	-	T	F
m_2	T	-	F
m_3	F	F	-

[The precise definition of cw_j is given in Equation (8)]. Hence, the argument $SW_j - cw_j$ for the function f represents the surplus weight of the material. The function $f(x)$ is defined so that small surplus x (typically $x < 5$) leads to a large penalty. In our parameter settings, we set $c_1 = 100$, $c_2 = 0.3$, and $c_3 = 0.05$. **Figure 2** shows the graph.

Constraint (5) ensures that consumed weight does not exceed the weight of the available material.

Constraint (6) is a *packing constraint*, which restricts the combinations of allocations of a material. Each piece of material is associated with a symmetric function $g : M \times M \rightarrow$ {TRUE, FALSE}, as shown in **Table 1**. When we allocate a piece of material j to orders by using two matches $m_1, m_2 \in M$, m_1 and m_2 must satisfy $g(m_1, m_2) =$ TRUE. When we allocate some material into three or more matches, each pair of matches must satisfy the function g. The packing constraint is due to the layout of the factory. Orders are associated with a unique route to fulfill the order according to its specifications. If orders with different routes are packed in the same batch of material, it may be impossible to cut the materials for the orders.

Matches

Equations (7) and (8) apply to matches. Each $(i, j) \in M$ ($\subseteq O \times S$) is associated with the following properties: cost C_{ij} (≥ 0), trimming loss TL_{ij} ($0 < TL_{ij} \leq 1$), and process yield PY_{ij} ($0 < PY_{ij} \leq 1$). The cost C_{ij} is a parameter for the objective function. A match with a high C_{ij} is favored for allocation. $TL_{i,j}$ and $PY_{i,j}$ are for the yield loss calculation.

Equation (7) defines the objective value for the piece of material j. The objective value for each piece of material j is defined by the product of the cost C_{ij} and the allocated weight of the match aw_{ij}.

Equation (8) defines the consumed weight cw_j of a material j. When we allocate material to orders, two types of losses arise: Loss for process yield and trimming loss [Figure 1(b)]. The loss for process yield always arises when we allocate material to orders. (If a piece of material is not allocated to any orders, there is no loss for process yield.) The term $SW_j PY_{ij}$ in Equation (8) represents the loss for the process yield. The process yield PY_{ij} is set so that $PY_{i_1,j} = PY_{i_2,j}$ holds if $g[(i_1, j), (i_2, j)] =$ TRUE for $(i_1, j), (i_2, j) \in M$. The trimming loss arises when the width of the material is larger than the required width of the order. The term aw_{ij}/TL_{ij} in Equation (8) represents the trimming loss for allocating match (i, j) with the weight of aw_{ij}.

In the MAP, the objective is to maximize the value of the objective function. Because both the profit PF_i of each order and the profit PF_j of each piece of material are relatively larger than the cost C_{ij} of each match, we obtain the highest score when the total allocated weight to an order i is equal to the target weight TW_i.

Because the formulation of the MAP contains nonlinear functions, we cannot use mixed-integer programming (MIP) solvers, such as ILOG CPLEX**. Therefore, we have designed a heuristic algorithm for this problem.

Basic algorithm

In this section we give an overview of our algorithm. In the first subsection, we describe our local search algorithm, in the second we show our acceleration techniques for the local search, and in the third we show our basic heuristics to reduce the surpluses.

Local search

As explained above in the Introduction, our algorithm is based on a VNS strategy. To construct an initial solution for the local search, we use a *random fit* strategy (a simple randomized variant of *first fit*). The random fit strategy permutes the matches in a random order, and our algorithm allocates the matches as far as possible one by one in that order.

For VNS, we constructed the following nine types of moves, which are divided into five groups. All of the moves are designed so that the objective score improves primarily by increasing the total weight of allocated materials.

Basic moves

In this group, we have two types of moves, a *simple move* and a *flip move*. In the simple move, shown in **Figure 3(a)**, we try to allocate a match when there is an allocatable match that is not allocated in the current solution. In the flip move, we remove a match and replace it with another match. **Figure 3(b)** illustrates a flip: We remove the match (A, X) and replace it with the match (A, Y). The allocated weight of the match is set to 0 in the removing step, and the allocated weight in the replacing step is as close as possible to the target weight of the order. The changes of the allocated weights are done similarly in the following moves.

Cyclic moves

In this group, we have two types of moves, a *two-cyclic move* and a *three-cyclic move*. In both moves, we exchange matches cyclically—i.e., some allocated materials are unallocated and replaced with other allocations. For example, a three-cyclic move is shown in **Figure 3(c)**: We remove the three matches (A, Z), (B, X), and (C, Y) and replace them with the three matches (A, X), (B, Y), and (C, Z). The two-cyclic move is defined similarly but can also be regarded (less formally) as a swap.

Shift moves

In this group, we have two types of moves, a *one-shift move* and a *two-shift move*. In both moves we remove a match and replace it with another match, and this move triggers additional allocations. For example, a two-shift move is shown in **Figure 3(d)**: We remove the two matches (B, X) and (C, Y) and replace them with the three matches (A, X), (B, Y), and (C, Z). The one-shift move is defined in a similar way: We remove a match and replace it with two matches.

Combination moves

In this group, we have two types of moves, an *order-optimal* (*order-opt*) *move* and a *material-optimal* (*material-opt*) *move*. In the order-opt (material-opt) move, we remove all allocated matches with respect to a single order (material) and reallocate with the best allocations with respect to the order (material). Strictly speaking, we cannot find the best allocations among all of the candidates because there are too many candidates due to the complex objective function, even if the order or material has a few allocatable matches. Therefore, the reallocations are done using suboptimal allocations. For the order-opt moves, we simply assume that only one

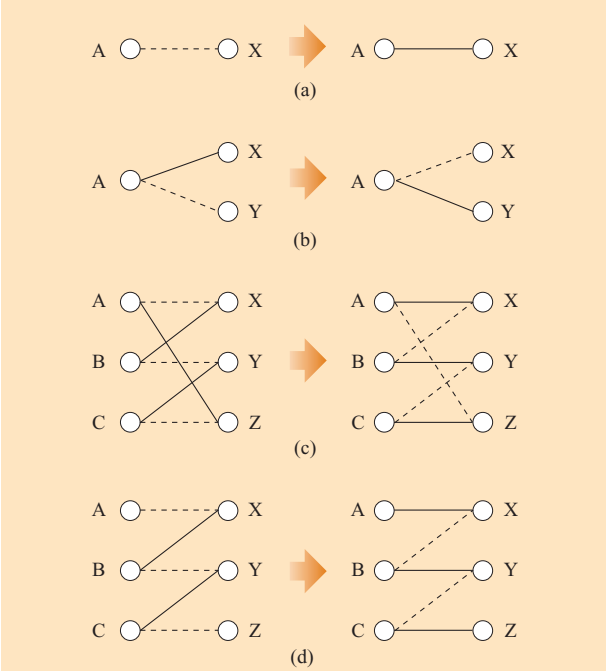

Figure 3

Examples of moves: (a) Simple move; (b) flip move; (c) three-cyclic move; (d) two-shift move.

piece of material can have a surplus in the best allocation and find suboptimal allocations under that assumption. Suppose that the order has k allocatable matches. Then, by that assumption, $k-1$ matches are assumed to be fully allocated or not allocated, and one batch of material can be partially allocated. Thus, there are at most $k(2^{k-1})$ candidates. Our algorithm finds the best allocation among them and replaces the old allocations with the best one. Similarly, for material-opt moves we assume that the orders, except for one, are allocated by their target weights, and the reallocations are done by using the suboptimal allocations.

Ejection chain search

The *ejection chain search* is the ninth move. This move is an extension of the shift moves; i.e., the ejection chain search performs a k-shift move for any integer k (see **Figure 4**). This type of move is used, for example, by Glover [3]. Because the ejection chain search is time-consuming, we first use one-shift and two-shift moves and then use the ejection chain search for $k \geq 3$.

To find an ejection chain, we use the concept of an improvement graph (Figure 4). An improvement graph is constructed from a current solution. Each node in the improvement graph corresponds to an allocated match in

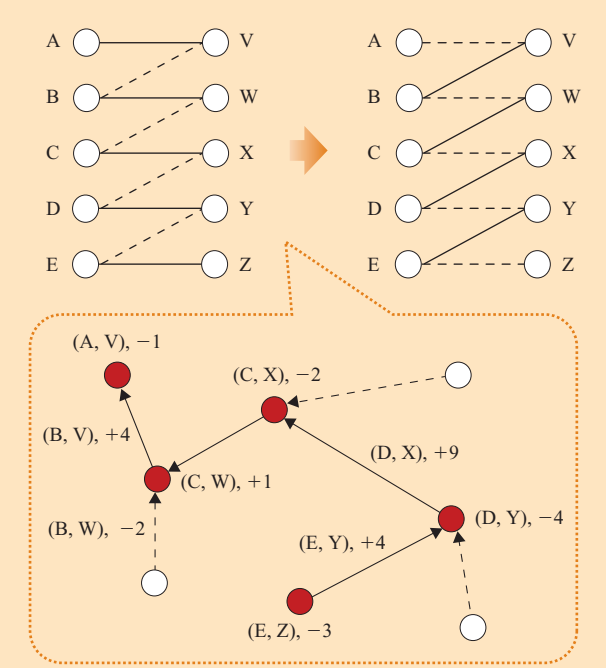

Figure 4

Ejection chain search and improvement graph.

the solution and is associated with a *profit* (the change in the objective value) when we remove the corresponding match. (Note that the profit is usually a negative value.) For example, node A shows that we reduce the objective value by 1 if we unallocate match (A, V). Each directed edge in the improvement graph corresponds to a match that is not allocated in the current solution. To illustrate, suppose that there are two allocated matches (A, V) and (B, W) and that a match (B, V) can be allocated if we remove the two matches (A, V) and (B, W). Then a directed edge corresponding to the match (B, V) is added to the improvement graph [the direction is from (A, V) to (B, W)]. The directed edge is associated with the profit when we allocate the match (B, V) in a situation in which the matches (A, V) and (B, W) are unallocated. For example, Figure 4 shows that we gain 2 in the objective score if we unallocate matches (A, V) and (B, W) and allocate match (B, V).

We search for a directed path or a cycle in the improvement graph that satisfies the following condition: Any two matches that correspond to two directed edges in the path or the cycle do not have a shared order or piece of material. If the path or the cycle satisfies this condition, the total profit associated with the path or the cycle is equal to the profit when we remove the matches that correspond to the nodes in the path or the cycle and replace the matches that correspond to the directed edges in the path or the cycle. For example, from the improvement graph in Figure 4, it can be seen that the allocations on the left are improved into the allocations on the right by using the path with solid arrows.

Therefore, all we have to do is to find a path or a cycle that satisfies the above condition in the improvement graph. In our implementation, this is done by a brute-force strategy. Though brute force is normally time-consuming, the algorithm finishes in a reasonable time when this search is applied to a current solution that all of the other moves cannot improve.

Variable neighborhood search

Overall, our algorithm applies the nine types of moves in the following order:

1. Simple move.
2. Flip move.
3. One-shift move.
4. Two-cyclic move.
5. Order-opt move.
6. Material-opt move.
7. Two-shift move.
8. Three-cyclic move.
9. Ejection chain search.

If a move successfully improves a current solution, our algorithm goes back to the first move (the simple move) and continues. If a move fails to improve a current solution, it tries the next move. Our algorithm finishes when all moves fail to improve the current solution. This order is approximately an increasing order of the neighborhood space of each move. Though this order may not be the best one, our experiments show that this order is the best among several candidates with respect to average execution time.

Acceleration

We now describe three techniques used to accelerate our local search algorithm.

Dealing with the unit weight constraints

Suppose there is an order whose minimum unit weight and maximum unit weight are a and b, respectively. When we check whether or not a given weight x satisfies the unit weight constraint of the order, the usual method is to check whether $an \leq x \leq bn$ holds by repeatedly incrementing the integer n by 1. Since it is easy to prove that $an \leq x \leq bn$ holds if and only if $x \leq \lfloor x/b \rfloor\, b$ holds,[1] we check the unit weight constraint with this inequality,

[1] In this paper, $\lfloor x \rfloor$ denotes the greatest integer not more than x and $\lceil x \rceil$ denotes the least integer not less than x.

which makes our algorithm efficient. If the weight x does not satisfy the unit weight constraint, it is also easy to show that $\lceil x/a \rceil\ a$ is the minimum weight not less than x that satisfies the unit weight constraints, and that $\lfloor x/a \rfloor\ b$ is the maximum weight not more than x that satisfies the unit weight constraint. These facts are also used in our algorithm.

Don't-look bit

We used the don't-look bit technique [4] to accelerate the local searches. This technique skips duplicates when trying to apply moves. In our implementation, we used this technique only to accelerate the local searches, though there are some implementations in which they accelerate local searches while sacrificing the quality of the solutions obtained in the local searches.

Grouping

Because of the packing constraint, some combinations of matches cannot be allocated at the same time. Therefore, to reduce the computation time for applying moves, we divide the matches associated with each batch of material into groups such that no pair of matches in different groups can be allocated at the same time. Since the function g for each batch of material can be regarded as an adjacency matrix for an undirected graph [i.e., if $g(m_1, m_2) = \text{TRUE}$, we can assume that the nodes m_1 and m_2 are connected by an edge], the grouping is done by identifying the connected components in the graph.

Reducing surplus

It is important to reduce the number of surpluses, especially the number of small surpluses. To avoid leaving a surplus, our basic algorithm uses a simple heuristic that tries to satisfy orders not only by their target weights but also by using the weights without surplus material. For example, suppose that a material with a weight of 5 tons is allocatable to an order that requires 4.5 tons. We normally try to allocate the material by the weight of 4.5 tons, but we also try to allocate the material with the weight of 5 tons if the order has enough allowance (i.e., the maximum allocatable weight for the order is at least 5 tons). Because a small surplus leads to a bad objective value in the objective function, the allocation of 5 tons is likely to be selected. This simple heuristic significantly reduces surplus materials.

TVIP local search

The heuristic for surplus reduction in the previous section is not fully satisfactory. There still remain a certain number of surplus pieces of material that could be further reduced. **Figure 5** illustrates an example in which the order A requires 14.3 tons and the order B requires 4.2 tons. Suppose that the minimum and maximum unit

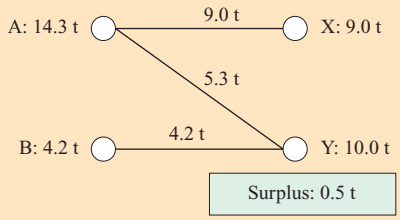

Figure 5

Example of an order with an unallocated surplus.

weights of the order A are 1.7 tons and 1.9 tons, and that the minimum and maximum unit weights of the order B are 2.0 tons and 2.3 tons. For this instance, the heuristic in the previous section fails to allocate the remaining surplus, while we can eliminate that surplus by allocating the material for order A with 5.6 tons and for order B with 4.4 tons. Therefore, we propose a move to deal with this problem which assuredly determines whether or not we can eliminate a surplus.

TVIP

We designed a move called the TVIP neighborhood search. In the TVIP, we release all allocations with respect to a batch of material with a surplus and reallocate the material without the surplus. The reallocation is done by trying all of the combinations of allocatable matches, but it is not trivial, even when the number of candidate matches is small. This move is inserted after the three-exchange move in the VNS.

Usually only a few batches of materials are split into more than two orders, so we focus on allocating the batch of material to at most two orders in the TVIP. It is trivial to check whether the material can be allocated to a single order without any surplus. Therefore, we focus on determining whether or not a batch of material can be allocated to two orders.

Suppose that we are given two orders A and B as well as a piece of material that has a weight of w tons and is allocatable to both orders. Suppose that the maximum unit weights of the orders A and B are u_{au} and u_{bu}, the minimum unit weights of the orders A and B are u_{al} and u_{bl}, and the maximum allocatable weights of the orders A and B are a_u and b_u, respectively. Let $a_l = 0$ and $b_l = 0$. (To simplify the following arguments, we assume that the losses for process yield and trimming do not arise when we allocate matches. We can show that TVIP works, even if we remove this assumption.)

Determining whether the material can be allocated to the orders without a surplus is equivalent to determining

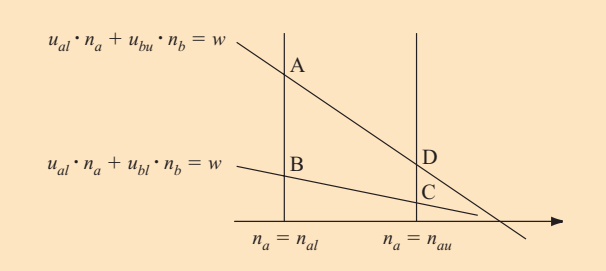

Figure 6

Feasible region for the integer programming problem of Equation (10).

whether there is a feasible solution subject to the following conditions:

$$x_a n_a + x_b n_b = w\,,$$

$$a_l \le x_a n_a \le a_u\,, \tag{9a}$$

$$b_l \le x_b n_b \le b_u\,, \tag{9b}$$

$$u_{al} \le x_a \le u_{au}\,, \tag{9c}$$

$$u_{bl} \le x_b \le u_{bu}\,, \tag{9d}$$

where

$$x_a,\ x_b \in R^+$$

and

$$n_a,\ n_b \in Z^+.$$

We do not specify objective functions in the conditions (9) because we are concentrating only on reducing the number of surpluses in this section, and every feasible solution has no surplus.

In the conditions in (9), x_a, x_b, n_a, and n_b are variables that respectively represent a unit weight for order A, a unit weight for order B, the number of units for order A, and the number of units for order B, respectively. Hence, $x_a n_a$ and $x_b n_b$ represent the weights of allocations for order A and order B.

Finding allocations without any surplus does not always lead to a better solution with respect to the objective function. However, because of the importance of reducing the number of small surpluses, finding allocations with no surplus usually leads to a better solution with respect to the objective function.

Equivalence with TVIP

Though finding a feasible solution of the conditions in (9) is not trivial, there is an efficient algorithm for finding a feasible solution. Suppose that the conditions have a feasible solution x_a, x_b, n_a, n_b. Then there exists another solution x_a', x_b', n_a', n_b' such that at least one of the eight equalities in (9a–d) holds (by perturbing x_a and x_b). When any of the four equalities in (9a) or (9b) holds, it is easy to find a feasible solution for all of the conditions in (9). For example, when we assume that there is a feasible solution such that $x_a' n_a' = a_l$, we only have to check whether or not the order B can be allocated with the weight $w - a_l$, which is a trivial check. The other three cases can be tested in a similar way. When any of the four equalities in (9c) or (9d) holds, we can also find a feasible solution efficiently. To prove this, first we show that there is an equivalent TVIP formulation for each case. If $x_a' = u_{al}$, the conditions in (9) are equivalent to the following TVIP problem, subject to

$$\begin{cases} x_a n_a + x_b n_b = w\,, \\ a_l \le u_{al} n_a \le a_u\,, \\ b_l \le w - u_{al} n_a \le b_u\,, \\ u_{al} n_b \le w - u_{al} n_a \le u_{bu} n_b\,, \\ n_a,\ n_b \in Z^+. \end{cases} \tag{10}$$

For the other three cases, we find similar TVIP problems. Thus, all we have to do is solve a TVIP problem (10) efficiently.

Algorithm for TVIP

The feasible region for the integer programming problem of (10) is depicted by the region ABCD in **Figure 6**, where

$$n_{al} = \max\left\{a_l/u_{al},\ (w - b_u)/u_{al}\right\}$$

and

$$n_{au} = \min\left\{a_u/u_{al},\ (w - b_l)/u_{al}\right\}.$$

Finding a feasible solution for the integer programming of (10) is equivalent to finding a lattice point (a point with integer coordinates) in that region.

It is easy to find a lattice point on line segments by using the following lemma (see Theorems 2–4 on p. 24 of [8]).

Lemma 1

The linear Diophantine equation $ax + by = c$ has a solution if and only if c is divisible by d, where $d = GCD(a, b)$ (the greatest common divisor of a and b). Furthermore, if (x_0, y_0) is a solution of this equation, the set of solutions of the equation consists of all integer pairs (x, y), where

$$x = x_0 + \frac{b}{d}k,\ y = y_0 - \frac{a}{d}k \quad (k = \cdots, -2, -1, 0, 1, 2, \cdots).$$

By using the extended Euclidean algorithm [9], $GCD(a, b)$ as well as the integers x_0 and y_0 such that

$ax_0 + by_0 = GCD(a, b)$ can be computed. Thus, by using Lemma 1, it is easy to check whether or not there exists a lattice point on a given segment.

To find a lattice point inside the above region, we use a binary search (using vertical lines to divide the region). In order to do the binary search, we need an algorithm to determine whether or not a lattice point is in a given region. It is sufficient if there is an algorithm for counting the number of lattice points in a given region. The following is an efficient algorithm for counting lattice points in right triangles:

Algorithm calcN(a, b, c)
  a, b, and c are positive integers such that $a \geq b$
begin
  $m := \lfloor c/a \rfloor$;
  if $a = b$ **then**
    return $m(m-1)/2$;
  else
    $k := \lfloor (a-1)/b \rfloor$; $h = \lfloor (c - am)/b \rfloor$;
    $t := km(m-1)/2 + mh$
    return calcN[b, $a - bk$, $c - b(km + h)$] + t;
  endif
end

This algorithm counts the number of lattice points in triangle

$$T(a, b, c) = \left\{(x, y) \in R^2 | ax + by \leq c,\ x > 0,\ y > 0\right\}.$$

Since the region can easily be divided into right triangles and rectangles, we can count the number of lattice points in the region efficiently. Thus, we can find a lattice point in the given region by using a binary search.

Remarks

Lattice-point-counting algorithms have been studied intensively for high-dimensional cases. For example, Barvinok gives an efficient algorithm [10], but it is theoretical and too complicated. (The algorithm is not explicitly described. To the best of our knowledge, there exists only one implementation of the algorithm, named `LattE` [11] because of its complication). For a two-dimensional case, Beck and Robins give an algorithm [5] that is much simpler, but still more complex than ours (the time complexity of our algorithm is the same as that of the algorithm of Beck and Robins). The correctness of our algorithm is given in [12]. Note that our algorithm deals only with integers, which contributes to its simplicity. We also note that Eisenbrand and Rote give an algorithm for solving TVIP [13], but their algorithm is also theoretical and not practical. (Their algorithm is fast when integers require long bits to encode them; i.e., integers are encoded far longer than 32 bits or 64 bits.)

Table 2 Sizes of instances.

Instance	*Orders*	*No. of materials*	*No. of matches*
A	2,000	4,000	50,000
B	3,000	5,000	50,000
C	2,000	4,000	50,000
D	3,000	5,000	50,000
E	2,000	6,000	50,000
F	3,000	6,000	50,000

It might seem that lattice points could be found by using simpler heuristics. A heuristic may suffice if there is a lattice point in the region, but in most cases the region does not contain any lattice point. This is because the solutions we consider here were supplied by the simple surplus-reduction heuristics in the previous section. Our algorithm is especially efficient if the region contains no lattice points, because in those cases we avoid the binary search.

Experiments

In this section, we show experimental results for our algorithm. All tests were executed on a PC workstation, an Intel Pentium** 4 running at 3.0 GHz, with 1 GB RAM using Microsoft Windows Server** 2003. The maximum time for execution was set to 3,600 seconds (one hour). This limitation is not too long for our client because the program runs at night.

To test our algorithm, we used six instances, sized as shown in **Table 2**. The differences among the six instances are the numbers of orders and materials. All of the instances were generated randomly in the following manner.

First, all matches were randomly generated. Every order was randomly divided into three groups (I, II, and III), and the packing constraints in the materials were determined so that orders in the same group could be allocated at the same time and orders in different groups could not be allocated at the same time.

The target weights of each order were randomly chosen from the range of 2.0–12.0 tons. The maximum total allocatable weights for each order were set to 1.2 times the target weight of the order. The minimum unit weight for each order was randomly chosen from 0.5 tons to 0.9 times the target weight of the order. The maximum unit weight for each order was randomly chosen from 0.5 tons to 1.5 times the minimum unit weight of the order. The profit per weight for each order was randomly chosen from the range [0–500].

The weight of each batch of material was randomly chosen from the range of 12.0–18.0 tons. The profit per

Table 3 Experimental performance results for local search.

Instance	*Objective value of initial solution*	*Average objective value after local search*	*Average time* (s)	*Gap* (%)	*Objective value of Kalagnanam et al. [7]*
A	756,683	7,644,174	269	0.16	5,650,904
B	777,064	11,170,235	391	0.26	8,471,049
C	764,954	7,639,779	248	0.16	5,761,081
D	872,737	11,203,976	359	0.16	8,579,216
E	785,952	7,629,990	196	0.06	5,826,122
F	892,738	11,214,853	285	0.15	8,662,206

Table 4 Surplus reduction with and without TVIP. Average execution times are shown in parentheses.

Instance	*Average number of small surpluses without TVIP* (s)	*Average number of small surpluses with TVIP* (s)	*Average reduction of surpluses* (%)	*Average increase in time* (%)
A	310.6 (272)	306.6 (269)	2.39	−2.10
B	435.0 (390)	427.0 (391)	1.84	0.26
C	309.6 (249)	307.2 (248)	0.78	−0.41
D	479.8 (343)	464.4 (359)	3.21	4.66
E	331.2 (193)	326.4 (196)	1.45	1.55
F	472.2 (286)	462.8 (285)	2.00	−0.35

weight for each material was randomly chosen from the range 0–500. The cost for each material was set to 1.

The cost of each match was set to 0. The trimming loss for each match was randomly chosen from the range [0.90–1.00]. The process yield of each match was set according to the group of the order of the match. The process yield was set respectively to 1.00, 0.98, and 0.96 for the group of orders I, II, and III.

We first tested the performance of our local search; **Table 3** shows the results. As the initial solutions were made by the random fit algorithm explained above in the section on local search, we executed our algorithm five times for each instance. The results in Table 3 are averaged over the five executions.

The table shows that our algorithm runs within the specified times and improves the objective values considerably from the initial solutions. The gaps show that objective values differ within only 0.3% after local searches. These results show that our local search algorithm works well and produces near-optimal solutions. The results of Kalagnanam et al. are for reference only. Because their algorithm was not designed for our model, a direct comparison cannot be made with this data, and we cannot conclude that our algorithm is superior to theirs from these experiments alone.

We used a different experiment to show the performance of our TVIP technique for surplus reduction. We executed our algorithm with and without TVIP five times each. **Table 4** shows the results. For each instance, it contains the average number of small surpluses without TVIP, the average number of small surpluses with TVIP, the average reduction ratio of surpluses, and the average ratio of increase of execution times. The numbers in parentheses are the average execution times.

Table 4 shows that TVIP succeeds in reducing the number of small surpluses without significantly increasing the computation time, which shows the effectiveness of applying TVIP in addition to the heuristics for surplus reduction in the basic algorithm. While the reduction is roughly 0.5–3.5 percent on random data in the experiments, TVIP reduces the number of small surpluses 5–10 percent on real data, which yields considerable profit to a steelworks.

We conducted another experiment to compare TVIP and the maximum-flow-based heuristics of Kalagnanam et al. [7], in which we replaced TVIP in our algorithm with their heuristics. However, the results showed that their heuristics did not improve either solution; that is, it produced the same solutions as our algorithm without TVIP. This shows that their heuristics may be effective

for reducing the total number of surpluses but are not as effective for reducing the number of small surpluses.

Concluding remarks

We constructed an algorithm based on variable neighborhood search for the MAP, which has resulted in considerable cost savings in a real steelworks. In particular, our TVIP technique has contributed to reducing the number of small surpluses of material without significantly increasing the computation times.

TVIP can be applied to other problems in real applications that cannot be modeled as well-known simplified models. Such problems can essentially include TVIP problems as subproblems. Even if the integer programming is associated with an objective function, the binary search in TVIP can find the best solution.

Our future work is to generalize our TVIP technique in order to adapt it to three or more variable integer programming problems. This generalization will depend on a practical and efficient algorithm for solving fixed-variable integer programming problems or a practical and efficient algorithm for lattice-point counting in fixed-dimensional polytopes (higher-dimensional analogs of polygons and polyhedrons).

Acknowledgments

We thank the IBM workers who made a dedicated effort to deliver our algorithm to the client. We also thank our colleagues, especially Takayuki Osogami, for their useful comments on this paper.

**Trademark, service mark, or registered trademark of ILOG Inc., Intel Corporation, or Microsoft Corporation in the United States, other countries, or both.

References

1. G. T. Ross and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Math. Program.* **8**, No. 1, 91–103 (1975).
2. P. Hansen and N. Mladenovic, "An Introduction to Variable Neighborhood Search," *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Editors, Kluwer Academic Publishers, Boston, MA, 1999, pp. 433–458.
3. F. Glover, "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," *Discrete Appl. Math.* **65**, No. 1, 223–253 (1996).
4. J. L. Bentley, "Fast Algorithms for Geometric Traveling Salesman Problems," *ORSA J. Computing* **4**, No. 4, 387–411 (1992).
5. M. Beck and S. Robins, "Explicit and Efficient Formulas for the Lattice Point Count in Rational Polygons Using Dedekind–Rademacher Sums," *Discrete & Computational Geometry* **27**, No. 4, 443–459 (2002).
6. J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee, "Inventory Matching Problems in the Steel Industry," *Technical Report RC-21171*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1998.
7. J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee, "The Surplus Inventory Matching Problem in the Process Industry," *Oper. Res.* **48**, No. 4, 505–516 (2000).
8. G. E. Andrews, *Number Theory*, Dover Publications, Inc., New York, 1994.
9. T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
10. A. I. Barvinok, "Computing the Ehrhart Polynomial of a Convex Lattice Polytope," *Discrete & Computational Geometry* **12**, No. 1, 35–48 (1994).
11. J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida, "Effective Lattice Point Counting in Rational Convex Polytopes," *J. Symbol. Comput.* **38**, No. 4, 1273–1302 (2004).
12. H. Yanagisawa, "A Simple Algorithm for Lattice Point Counting in Rational Polygons," *Technical Report RT-0622*, Tokyo Research Laboratory, IBM Japan, Ltd., Yamato-shi, Kanagawa-ken, 242-8502, Japan, 2005.
13. F. Eisenbrand and R. Rote, "Fast 2-Variable Integer Programming," *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization*, Utrecht, The Netherlands, 2001, pp. 78–89.

Received September 19, 2006; accepted for publication November 28, 2006; Internet publication May 18, 2007

Hiroki Yanagisawa *IBM Research Division, Tokyo Research Laboratory, 1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa 242-8502, Japan (yanagis@jp.ibm.com)*. Mr. Yanagisawa received B.S. and M.S. degrees in computer science from Kyoto University, where he specialized in approximation algorithms on combinatorial optimization problems. In 2003 he joined the IBM Research Division, where he works on optimization problems in manufacturing industries and logistics. Mr. Yanagisawa's research interests include algorithms in operations research and computer science.