Solution of a tinned iron purchasing problem by Lagrangean relaxation

B. Dorhout

Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 Enschede, Netherlands

Received September 1989; revised March 1990

Abstract

A tin factory obtains its material from steel works. This consists of sheets of tinned iron which may have very diverging specifications with respect to length, width, thickness, and thicknesses of tinfoils. Prices per unit of volume vary with width and thickness. For large quantities of the same size discounts are given. As a consequence of the price structure it is often advantageous for the factory to order sheets of larger sizes than needed and to resell the leftover pieces as scarp. The question is which sizes and quantities one should order if one wishes to minimize total purchase cost. This problem is formulated as a combinatorial optimization problem that is solved by Lagrangean relaxation and subgradient techniques.

Keywords: Quantity discounts; Assortment; Combinatorial optimization; Lagrangean relaxation

1. Introduction

In this paper we describe a purchasing problem originating from a large tin factory in the Netherlands, and the way we solved it. The problem is as follows.

Raw material for tins consists mainly of rectangular iron sheets, in most cases covered by a tin layer at one or both sides. It is purchased from steel works. As tins are used for the storage of very different products, like, for instance, meat, beans, paint, lubricating oil, beer, and hair spray, the clients ask for very different kinds of tins with different properties and sizes.

For each order received the laboratory of the factory calculates how much rectangles are needed, which sizes they should have, how thick tin layer(s) should be, and what their other specifications should be, for instance which temper and hardness. This is done in such a way that steel works can deliver the rectangles exactly as specified. That means in particular:

- thicknesses of tin layers are selected from a number of discrete values;
- for thicknesses of iron sheets (without tin) one can choose from 0.15, 0.16, ..., 0.50 mm;
- lengths and widths are allowed to have arbitrary values.

One wishes to minimize purchase costs for a fixed production program. This can be achieved by earning optimal discounts, exploiting the price structure described below.

Quantity discounts are given for large quantities of identical sheets. These can be realized by ordering sheets with thicker tin layers than needed, and/or sheets which are at most 0.01 mm thicker, and/or sheets of greater length and/or width. For each order only identical sheets can be used. For quantities
from 50 to 100 tons a 1% discount is given, from 100 to 150 tons this is 1.5%, and for more than 150 tons it is 2%. Orders for less than 25 tons are considered to be undesirable.

For each thickness the factory has to pay a fixed price per m$^2$ of sheet. Standard prices are in force for widths in the range from 700 to 750 mm. For widths in the range from 750 to 800 mm they are decreased with a fixed amount, in the range from 800 to 850 mm prices decrease with a greater amount, and so on. For widths smaller than 700 mm there are, in the same way, intervals with higher fixed prices per m$^2$.

Tin layers cost a fixed amount per m$^2$, dependent on their thickness.

If sheets are cut off to the size needed, scrap is sold at a fixed amount per ton, which is independent of the thickness of tin layers.

Changes in temper, hardness or other specifications not mentioned are not allowed. So for each combination of these other specifications one can solve the same kind of problem separately. As in the production process the rolling direction plays an important role it is not allowed to interchange length and width of a sheet. Also it is not allowed to interchange the upper and lower side of a sheet. If these interchanges had been allowed, the problem could have been solved in about the same way as described below, but at the expense of a great increase of the size of the mathematical model.

We formulated a combinatorial programming model of the problem described above and made a program for its solution on a micro computer. This program is not only used for monthly ordering and forecasting, but also for negotiation purposes: in negotiations with clients one is able to calculate costs of wishes with respect to types and quantities of tins.

In Section 2 we describe our model and in Section 3 an algorithm for solving it. Some experience with the algorithm and an extension of the program are formulated in Section 4.

2. Mathematical model

The solution of the purchasing problem should indicate the sizes and quantities of the sheets one has to order. For simplicity we use the word *size* for the 5-tuple length, width, thickness, thickness of upper tin layer, thickness of lower tin layer. As a first step in formulating the mathematical model we construct a set of sizes which contains the set of optimal sizes. From now on we will use the terms *needed size* for a size which is needed for production and *order size* for a size which can be ordered.

There is a simple procedure of gradually generating the desired set of order sizes from the set of needed sizes. For a description we use the notion of join. In general the join $c = (a, b)$ of two n-tuples of reals $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ is defined as $c = (\max(a_1, b_1), \ldots, \max(a_n, b_n))$. We use this definition for our sizes, with $n = 5$. Then the principle of the procedure can be described as follows.

1. Denote the set of needed sizes by $B$ and make a copy $E$ of $B$.
2. Generate the set of joins $(b, e)$ of all $b \in B$ and $e \in E$. Add this set to $E$ and remove duplicates.
3. Compare each size in $E$ with the size obtained by replacing its width with the lower limit of the next higher width interval with lower price per m$^2$ (if this exists). If the original size is more expensive replace it by the modified one. Remove duplicates.

Notice that step 2 will never be repeated more than three times because by then all possible joins of 5-tuples are included in the extended set $E$. In the computer program that solves the problem we used in principle this procedure, with the exception that step 4 is executed partially within step 2.

Before formulating the problem we first introduce some notation.

Define the sets:

$I = \text{set of indices of order sizes},$

$J = \text{set of indices of needed sizes},$

$D = \{1, 2, \ldots, g\} = \text{set of indices of quantity discount classes},$

$R_i = \text{subset of needed sizes } j \in J \text{ that can be replaced by order size } i, i \in I,$

$S_j = \text{subset of order sizes } i \in I \text{ that can replace needed size } j, j \in J,$

the constants:

$\ell_d = \text{lower bound, in tons, for quantity discount class } d, d \in D, \text{ with } \ell_d \leq \ell_{d+1}, d = 1, \ldots, g - 1,$

$\ell_{g+1} = \text{upper bound, in tons, for the sum of all needed quantities},$
\[ c_{id} = \text{price, in guilders per ton, of sheets of size } i \text{ in quantity discount class } d, \ i \in I, \ d \in D, \]
\[ a_{ij} = \text{total weight of sheets of size } i, \text{ in tons, needed to replace sheets of size } j, \text{ if need for size } j \text{ is covered by sheets of size } i, \ j \in R_i, \ i \in I, \]
\[ p_{ij} = \text{yield, in guilders, of all scrap from sheets of size } i, \text{ as far as they are used to replace sheets of size } j, \ j \in R_i, \ i \in I, \]

and the variables:
\[ x_{id} = \text{weight, in tons, of sheets } i \text{ to purchase in quantity discount class } d, \ i \in I, \ d \in D, \]
\[ u_{id} = \begin{cases} 1 & \text{if sheets } i \text{ are purchased in quantity discount class } d, \\ 0 & \text{otherwise}, \end{cases} \ i \in I, \ d \in D, \]
\[ y_{ij} = \begin{cases} 1 & \text{if sheets } i \text{ replace sheets } j, \\ 0 & \text{otherwise}, \end{cases} \ j \in R_i, \ i \in I. \]

The problem can now be stated as:

\begin{align*}
\text{minimize} & \quad z = \sum_{i \in I} \sum_{d \in D} c_{id} x_{id} - \sum_{i \in I} \sum_{j \in R_i} p_{ij} y_{ij} \\
\text{subject to} & \quad \sum_{d \in D} x_{id} \geq \sum_{j \in R_i} a_{ij} y_{ij}, \quad i \in I \\
& \quad \ell_d u_{id} \leq x_{id} \leq \ell_{d+1} u_{id}, \quad i \in I, \ d \in D \\
& \quad \sum_{d \in D} u_{id} \leq 1, \quad i \in I \\
& \quad \sum_{i \in S_j} y_{ij} = 1, \quad j \in J \\
& \quad y_{ij} = 0 \text{ or } 1, \quad j \in R_i, \ i \in I \\
& \quad u_{id} = 0 \text{ or } 1, \quad i \in I, \ d \in D. \quad (1), (2), (3), (4), (5), (6), (7)
\end{align*}

In this model the objective function (1) equals purchase costs minus returns from scrap. Relation (4) assures that for each \( i \in I \) sheets are purchased in at most one quantity discount class. While (3) guarantees that for each \( i \) at most one variable \( x_{id} \) has a positive value, within the correct limits, (2) forces this value to be large enough. Restriction (5) says that exactly one order size will be used for each needed size \( j \in J \).

We shall refer to the problem defined by (1), . . . , (7) as to problem (P).

3. Solution of the model

We obtained very good solutions to problem (P) by the application of a Lagrangean relaxation method, as described by Geoffrion (1974): After the introduction of a vector \( v \) of Lagrangean multipliers \( v_j, j \in J \), associated to constraints (5), we define the function

\[ f(v) = \min \left\{ \sum_{i \in I} \sum_{d \in D} c_{id} x_{id} - \sum_{i \in I} \sum_{j \in R_i} p_{ij} y_{ij} + \sum_{j \in J} v_j \left( 1 - \sum_{i \in S_j} y_{ij} \right) \mid (2), (3), (4), (6), (7) \right\} \]

which for each \( v \) gives a lower bound to the optimum value of (P). This function was maximized by a subgradient method as described by Held et al. (1974), with the improved step size given by Camerini et al. (1975). For the evaluation of \( f(v) \) the relaxed integer linear programming problem at the right hand side of (8) has to be solved. We shall denote it as (PR).

If the solution to (PR) satisfies (5), then an optimal solution to (P) is obtained. In general, however, this is not the case. Then an iterative method is executed. During the iterations an upper bound \( z^* \) and
a lower bound $z_*$ for the optimal value of $z$ are available. At the start of the iterations we take $z^* = \infty$ and $z_* = -\infty$. These values are updated every time better values are obtained. Knowledge of $z^*$ and $z_*$ gives $(z^* - z_*) / z_*$ as an upper bound for the relative deviation of $z^*$ from the optimal value. As for most problem instances this bound decreases fast during the first iterations and later on not much progress is made the algorithm is stopped as soon as $(z^* - z_*) / z_* < \varepsilon$, an appropriate chosen small number, or after a prescribed number of iterations $k_0$.

A short global description of the method is as follows:

**Step 0.** $k \leftarrow 0$; Calculate a start vector $v^0$;

**Step 1.** (start of new iteration) $k \leftarrow k + 1$;
- calculate $z = f(v^{k-1})$, giving an optimal vector $y = \bar{y}$ to (PR);
- if $z > z_*$ then $z_* \leftarrow z$;

**Step 2.** derive from $\bar{y}$ a feasible solution $(\bar{x}, \bar{u}, \bar{y})$ to (P) with objective function value $\bar{z}$;
- if $\bar{z}$ satisfies (5), stop: $(\bar{x}, \bar{u}, \bar{y})$ is optimal for (P);
- if $\bar{z} < z^*$ then $z^* \leftarrow \bar{z}$; store $(\bar{x}, \bar{u}, \bar{y})$ as incumbent solution to (P);
- if $(z^* - z_*) / z_* < \varepsilon$ or $k = k_0$ then stop: the incumbent solution is the best solution found;

**Step 3.** derive from $\bar{y}$ a step $\bar{s}$ in $v$-space, set $v^k \leftarrow v^{k-1} + \bar{s}^k$, and go to Step 1.

We shall now give a description of some details of the algorithm. For explanatory reasons we shall do this in the order of Step 1, followed by Steps 0, 3, and 2 respectively.

**Step 1**

Working out the definition of $f(v)$ it appears that the problem that has to be solved decomposes into a number of smaller problems, which turn out to be knapsack problems. One finds

$$f(v) = \min \left\{ \sum_{i \in I} \sum_{d \in D} c_{id} x_{id} - \sum_{i \in I} \sum_{j \in R_i} (v_j + p_{ij}) y_{ij} + \sum_{j \in J} v_j \right\}$$

$$= \sum_{j \in J} v_j + \min \left\{ \sum_{i \in I} \sum_{d \in D} c_{id} x_{id} - \sum_{i \in I} \sum_{j \in R_i} (v_j + p_{ij}) y_{ij} \right\}$$

subject to:

$$\sum_{d \in D} x_{id} \geq \sum_{j \in R_i} a_{ij} y_{ij}, \quad i \in I$$

$$\ell_d u_{id} \leq x_{id} \leq \ell_{d+1} u_{id}, \quad i \in I, \quad d \in D$$

$$\sum_{d \in D} u_{id} \leq 1, \quad i \in I$$

$$y_{ij} = 0 \text{ or } 1, \quad j \in R_i, \quad i \in I$$

$$u_{id} = 0 \text{ or } 1, \quad d \in D$$

$$d \in D$$

$$\left(9\right)$$
where the last equation follows from the property that the constraint set can be partitioned into constraint sets for each \( i \in I \) separately. Then it follows easily from the definition of the \( u_{id} \)-variables that

\[
\begin{align*}
  f(v) &= \sum_{j \in J} v_j + \sum_{i \in I} \min_{d \in D} \left\{ \begin{array}{l}
    0, \min_{d \in D} \left\{ \sum_{d \in D} c_{id} z_{id} - \sum_{j \in R_i} (v_j + p_{ij}) y_{ij} \right\}, \\
    x_{id} \geq \sum_{j \in R_i} a_{ij} y_{ij}, \\
    \ell_d \leq x_{id} \leq \ell_{d+1}, \\
    y_{ij} = 0 \text{ or } 1, \quad j \in R_i
  \end{array} \right\}.
\end{align*}
\]

(10)

Inspection of this relation indicates that the upper bounds on the \( x_{id} \)-values can be omitted. In fact, \( X_{id} = \max(\ell_{d}, \sum_{j \in R_i} a_{ij} y_{ij}) \) in the optimal solution, since all \( c_{id} \) are positive, and \( x_{id} = \ell_{d+1} \) could only occur if \( \sum_{j \in R_i} a_{ij} y_{ij} = \ell_{d+1} \), but then \( x_{id+1} = \ell_{d+1} \) gives the same \( y \)-solution, with a lower value of the objective function, as a consequence of \( c_{id+1} < c_{id} \). Hence (10) can be replaced by

\[
\begin{align*}
  f(v) &= \sum_{j \in J} v_j + \sum_{i \in I} \min_{d \in D} \left\{ \begin{array}{l}
    c_{id} z_{id} - \sum_{j \in R_i} (v_j + p_{ij}) y_{ij} \right\}, \\
    \sum_{i \in I} y_{ij} = 0 \text{ or } 1, \quad j \in R_i
  \end{array} \right\}.
\end{align*}
\]

(11)

So two cases are compared: \( \sum a_{ij} y_{ij} \geq \ell_d \) and \( \sum a_{ij} y_{ij} \leq \ell_d \). In the first case \( x_{id} = \sum a_{ij} y_{ij} \), in the second case \( x_{id} = \ell_d \). Observe that in principle \( 2 * |I| * (|D| - 1) \) knapsack problems must be solved for the evaluation of \( f(v) \). In practice the value of \(|I|\) can be in the order of some hundreds and \(|D| = g\) is a small number, in real situations considered by us: 4.

From the second part of (11) it appears that for each \( i \in I \) separately only the solution of that knapsack problem from a set of \( 2 * g - 2 \) is needed, that gives the maximal contribution to the second term. By this property it is not necessary, in general, to solve for each \( i \in I \) all \( 2 * g - 2 \) corresponding knapsack problems to optimality. We start solving LP-relaxations, obtained by replacing the conditions \( y_{ij} = 0 \text{ or } 1 \) by \( 0 < y_{ij} < 1 \). These often have integer solutions and always give an upper bound for the value of the objective function. If all upper bounds are nonpositive, we are ready with the \( i \) considered. Otherwise we start solving exactly the knapsack problem with the highest upper bound. This gives us a lower bound for the contribution to \( f(v) \) of this \( i \). So it is not necessary any more to solve those knapsack problems for which the LP-relaxation upper bounds are lower than this lower bound.

The knapsack problems are solved by the algorithm of Martello and Toth (1977). This algorithm has the property that a substantial fraction of computing time is used in a preprocessing step in which the variables are sorted in the order of ratios of cost and weight coefficients. In our application of the
algorithm we exploit this property: it is easy to see that these orders are the same in all knapsack problems for all order sizes \( i \) with the same thickness.

**Step 0**

In constructing a start solution for the subgradient maximization of \( f \) we followed Geoffrion (1974). He proved that a good choice is the vector \( v^0 \) with components equal to the values of the simplex multipliers for (5) associated with the optimal solution to the LP-relaxation of (P). It is easy to see that this LP-relaxation boils down to

\[
\sum_{j \in J} \min \left\{ \sum_{i \in S_j} (c_{id}a_{ij} - p_{ij})y_{ij} \middle| \sum y_{ij} = 1 \right\}
\]

with \( d = \max(d \mid d \in D) \), which is independent of \( i \). So the components of \( v^0 \) are

\[
v^0_j = \min_{i \in S_j} (c_{id}a_{ij} - p_{ij}), \quad j \in J.
\]

In general the expression in (13) will be equal to \( c_{id}a_{ii} - p_{ii} = c_{id}1 - 0 = c_{id} \). Only if a sheet \( j \) is more expensive than a sheet of the same specifications but with a higher width interval, it will be different.

**Step 3**

If the \( y_{ij} \)-values \( y_{ij} \), found in Step 1, do not satisfy (5), then the vector \( h^k \) with components

\[
h^k_j = 1 - \sum_{i \in S_j} y_{ij}, \quad j \in J.
\]

is a subgradient of \( f \) in \( v^k \). The step \( s^k \) in the expression

\[
v^k \leftarrow v^{k-1} + s^k, \quad k = 1, 2, \ldots,
\]

is, according to Camerini et al. (1975),

\[
s^k = \mu^k s^k, \quad k = 1, 2, \ldots,
\]

with

\[
s^k = h^k - \begin{cases} 
\frac{\gamma (h^k)^T s^{k-1}}{\| s^{k-1} \|^2} s^{k-1} & \text{if } (h^k)^T s^{k-1} < 0 \text{ and } k > 1, \\
0 & \text{otherwise},
\end{cases}
\]

and

\[
\mu^k = \frac{z^* - f(v^{k-1})}{\| s^k \|^2},
\]

where \( \gamma \) and \( \rho \) are real numbers, with \( 1 \leq \gamma \leq 2 \), and \( 0 < \rho \leq 1 \).

**Step 2**

If the \( y_{ij} \)-values \( y_{ij} \) resulting from the optimal solution to (11) satisfy the relaxed constraints (5) they are optimal for (P). In this case the values of the other variables in the optimal solution to (P) are derived in the following way.

For each \( i \in I \) examine the contribution to the objective function of (11). If it is 0 then we must have

\[
x_{ij} = \tilde{x}_{ij} = 0, \quad u_{ij} = \tilde{u}_{ij} = 0 \quad \text{for all } d \in D,
\]

(19)
else we know which \( d = d(i) \) is associated with the optimal solution to \((PR)\) and

\[
\begin{align*}
    u_{id} = \bar{u}_{id} &= \begin{cases} 
        1 & \text{if } d = d(i), \\
        0 & \text{otherwise,} 
    \end{cases} & d \in D, \\
    x_{id} = \bar{x}_{id} &= \begin{cases} 
        \max \left( \sum_{j \in R_i} a_{ij} \bar{y}_{ij}, \ell_d \right) & \text{if } d = d(i), \\
        0 & \text{otherwise,} 
    \end{cases} & d \in D.
\end{align*}
\]

If the \( \bar{y}_{ij} \)-values do not satisfy \((5)\), then they are changed into values that do, and after that a feasible solution to \((P)\) is derived by means of formulas \((19)\), \((20)\), and \((21)\). If \( B_j = \{ i \in I \mid \bar{y}_{ij} = 1 \} \), then a solution to \((P)\) satisfies \((5)\), i.e. each needed size is assigned to exactly one order size, iff \( |B_j| = 1 \) for all \( j \in J \). In order to obtain a good feasible solution we try to assign the needed sizes \( j \) to those order sizes, which occur in the solution found in Step 1, and which produce small quantities of scrap.

In a preprocessing step to the algorithm we set up a list \( L \) of all elements of \( \bigcup_{j \in J} B_j \), arranged in order of ascending volumes of order sizes, with order sizes of equal volume arranged in order of ascending thickness of tin layers. Thus it will be favourable to assign a needed size \( j \) to the first size of \( L \) that belongs to \( S_j \).

Then the algorithm consists of the following steps:

**Step a.** Reduce every \( B_j \) with \( |B_j| > 1 \) to its elements in \( L \) with lowest index;

**Step b.** For all elements \( i \) of \( L \), in ascending order:

- for all \( j \) with \( B_j = \emptyset \) and \( j \in S_j \) set \( B_j \leftarrow \{ i \} \);
- if \( \sum_{j \in B_j} a_{ij} \ell_j < \ell_1 \) then set \( B_j \leftarrow \emptyset \) for all \( j \) with \( B_j = \{ i \} \);

**Step c.** If any \( j \) with \( B_j = \emptyset \) is left then:

- determine the join \( i^+ \) of all such \( j \), and set \( B_j \leftarrow \{ i^+ \} \) for these \( j \);

**Step d.** For every \( i \in \bigcup_{j \in J} B_j \):

- determine the join \( i^* \) of all \( j \) with \( B_j = \{ i \} \);
- if \( i \neq i^* \) then for all \( j \) with \( B_j = \{ i \} \) set \( B_j \leftarrow \{ i^* \} \).

This algorithm apparently has the following properties:

- in Steps a and b only order sizes found in Step 1 are considered,
- after completion of step b there is no order size to which needed sizes are assigned with a total weight less than \( \ell_1 \),
- after completion of Step c all needed sizes are assigned to an order size,
- if a set of needed sizes is assigned to an order size that is more expensive than necessary then the elements of this set are reassigned to the cheapest possible enclosing order size.

**Implementation**

After the description of Steps 0 through 3 three details have to be elaborated yet: the choice of the values of \( \gamma \) in formula \((17)\) and \( \rho \) in formula \((18)\) and the stopping criterion. These were developed by experimentation with a computer program working on subsets of a set of needed sizes.

We started applying the rules suggested by Held and Karp (1971), i.e. we set \( \gamma = 0 \) and let decrease \( \rho \). This gave erratic results: with identical rules the \( f(v^k) \)-sequence was divergent for some problems and convergent to a suboptimal value for some other problems. With \( \gamma = 1.4, 1.5, \) and \( 1.6 \) these situations did not occur. It appeared that results with fixed values of \( \rho \) were not worse than results with decreasing values. As the best results were obtained with \( \rho = 1 \) this value was fixed in the computer program. Further we fixed \( \gamma = 1.5 \), as this value gave the fastest convergence, which is also in accordance with results by Camerini et al. (1975).
In searching for a stopping rule we checked how the quality of the solutions depends on the number of iterations. It turned out that in general after 50 iterations not much or no improvement in solutions occurred, irrespective of \(|J|\), |I| and |D|, which is in accordance with results reported by Gondran and Minoux (1984). So we decided to choose as stop parameters in Step 2: \(\varepsilon = 0.001\) and \(k_0 = 50\). With this rule the largest value of the upper bound \((z^* - z_*)/z_*\) for the relative error in \(z\) we ever obtained for our test set, was 0.007.

4. Practical use, and an extension

The algorithm was programmed as a procedure in Turbo Pascal that was embedded in a program for micro computers in which quantities and specifications needed are read from the factory’s data base. They can be changed and extended from the keyboard of the computer.

Although the program produced very good solutions to the problem as it was posed, it appeared that often the number of different order sizes indicated in the solution was larger than expected. As one prefers to have small numbers of sizes in stock, one asked us to extend the program in such a way that not only near optimal solutions would be obtained for our problem (P), but also least cost solutions would be generated with smaller numbers of order sizes. So, according to these wishes, a second phase was added that gives solutions to the problem with numbers of order sizes decreasing from the optimum to the minimum. (If the difference in thickness between the needed sizes is not greater than 0.01 cm then this minimum is 1.)

We assumed that, in general, order sizes used in the final solution to (P) would remain favourable order sizes for solutions in the second phase of the program. So initially we derived successively all second phase solutions from the current final solution to (P) by repeated application of the following simple rule: replace the current solution with the solution that can be obtained from it by deletion of that current order size that gives the smallest increase of the objective function after completion of the partial solution by Step 2.

As the results of this rule were rather bad we changed deletion of one used order size at a time into deletion of two order sizes and adding the join of them. This revised procedure takes some more time, but it gives much better results. With our test set it appeared that in most cases it was possible to decrease the number of different order sizes at a very low increase of cost, at least in the first iterations of the second phase.

Acknowledgment

The author is greatly indebted to Ir. H.M. Heemskerk from Consulting Office Oude Vrielink and Heemskerk for programming the whole input part of the program.

References


