ELSEVIER

European Journal of Operational Research 106 (1998) 317–335

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

The hot strip mill production scheduling problem: A tabu search approach

Leo Lopez [a,b], Michael W. Carter [a,*], Michel Gendreau [c]

[a] Department of Mechanical and Industrial Engineering, University of Toronto, 5 King's College Road, Toronto, Ont., Canada M5S 3G8

[b] Departamento de Administracion de Empresas, Universidad de las Americas-Puebla, A.P. 100, Santa Catarina Martir, 72820 Puebla, Mexico

[c] Département d'informatique et recherche opérationnelle, Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal, Que., Canada H3C 3J7

Received 1 April 1996; received in revised form 1 April 1997

Abstract

The Hot Strip Mill Production Scheduling Problem is a hard problem that arises in the steel industry when scheduling steel coil production. It can be modeled as a generalization of the Prize Collecting Traveling Salesman Problem with multiple and conflicting objectives and constraints. In this paper we formulate this problem as a mathematical program and propose a heuristic method to determine good approximate solutions. The heuristic is based on Tabu Search and a new idea called "Cannibalization". Computational results on production data from Dofasco are presented and analyzed. Comparison with actual production schedules indicates that the proposed method could produce significantly better schedules.

Keywords: Tabu search; Scheduling application; Steel production; Heuristics; Multicriteria optimization

1. Introduction

The operations in an integrated steel plant can be classified in two sections: primary and finishing. The primary operations produce steel products from raw material such as iron ore and coal. They are transformed in the blast furnace into liquid iron, in the melt shop into liquid steel, in the continuous caster area into solid large steel bars, and in the hot strip mill (HSM) into the final product, steel coils, that we will consider in this research (see Fig. 1). The finishing operations control the final specifications of the orders. Pickling eliminates surface oxidation on the coils, cold rolling takes care of ordered gauge, annealing restores ductility, tempering restores surface strength coil properties, and finally, galvanization, tin and prepainting accomplish the requested surface coating.

We focus our attention on the HSM area of the primary operations. This mill transforms large

* Corresponding author. Fax: +1 416 978 3453; e-mail: carter@mie.utoronto.ca.

0377-2217/98/$19.00
PII S0377-2217(97)00277-4

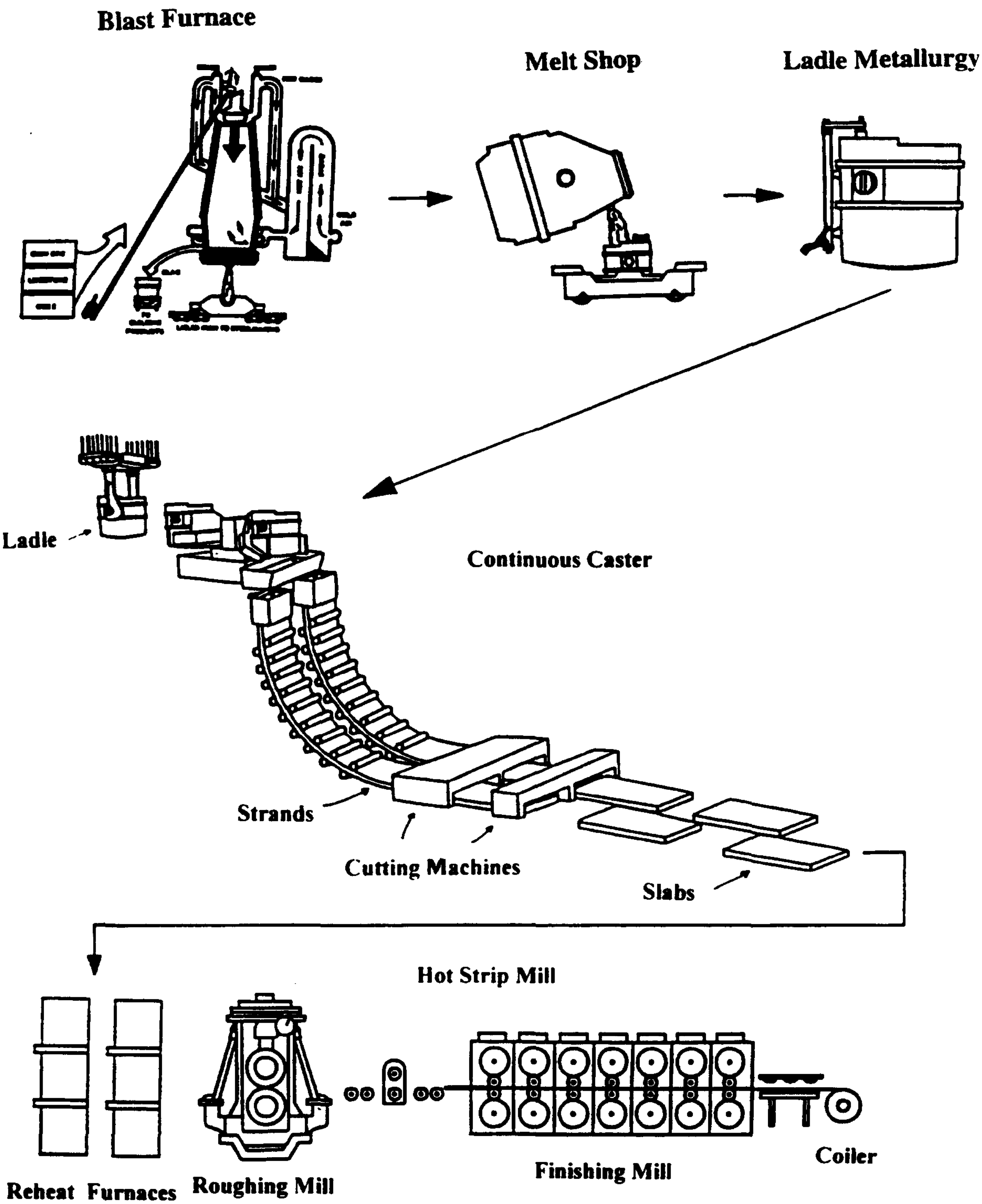

Fig. 1. Primary operations in the steel industry.

bars of steel called slabs into coils of steel sheets to be used in the manufacture of products such as automotive bodies or appliance shells. The Hot Strip Mill Production Scheduling Problem (HSMPSP) consists of the generation of production schedules for the HSM area. There are several conflicting objectives that have to be achieved subject to several limitations. It is desirable for the HSM to obtain a nice sequence of slabs with similar characteristics, to avoid set-up times and therefore maintain a smooth production process. On the other hand, high quality in products is an aim. It is important for another area of the plant (the heating area) to use the resources efficiently to avoid waste of energy, while another area of the plant (sales) expects to obtain final products on time to avoid delays in delivery. However, the company would like to have a high productivity rate. To make the problem more complicated, there are several kinds of constraints in the plant. A nice sequence of slabs for one area may be a non-desirable or impossible sequence for another area. The solution to this problem should balance those conflicting objectives and constraints.

The mathematical formulation of the HSMPSP that we present in this paper can be interpreted as a generalization of the Prize Collecting Traveling Salesman Problem (PCTSP), which is itself a generalization of the Traveling Salesman Problem (TSP), well known as an NP-hard problem. For this reason, we suspect that the HSMPSP is an NP-hard problem too (as defined in Section 2).

In this paper we approach a real HSMPSP for a Canadian steel company, Dofasco, in Hamilton, Ontario, Canada. Since the HSMPSP is hard to solve to optimality, we have developed a heuristic method based on the Tabu Search metaheuristic and obtained some results that will be discussed later. Section 2 describes the HSMPSP. Section 3 reviews previous work done by other researchers. Section 4 gives a formulation of the problem as a mathematical programming problem. Section 5 presents a heuristic algorithm based on the Tabu Search metaheuristic to approach the solution. Section 6 provides some computational experience. Finally, Section 7 presents some conclusions.

2. Problem description

To describe the HSMPSP at Dofasco in more detail, we have to describe the production process of steel coils. Liquid steel is produced in 300 ton-batches called *heats*. Each heat is made to satisfy a group of orders of steel of identical chemical composition (grade). Each heat produces about 16 long bars, called slabs, in the continuous caster area and all of them have the same grade. Sometimes all of the slabs are used to satisfy demands that already exist and sometimes they have to be stored. When the slabs are not already assigned to orders, they are stored in a remote yard until they are assigned to orders. If the slabs have already been assigned to orders, they may be sent either to a local yard where they will stay for at least 3 h before being processed, or directly to the heating area.

When the slabs leave the caster area, they have a temperature above 900°C and we say that the slabs are *hot*. When the slabs have been waiting in the local yard before being charged into the reheat furnaces, their temperatures fluctuate between 100°C and 800°C and they are said to be *warm*. When the slabs have been sent to the remote yard, their temperatures are ambient before being charged into the furnaces and we say that those slabs are *cold*. In any case the slabs are reheated in one of two reheat furnaces (RFs) in the heating area. Slabs must be between 1185° and 1250° to be processed on the HSM. The temperature of a slab right before being "charged" into the reheat furnaces is called the *charging temperature* and the temperature that a slab needs to reach to be processed in the HSM is called the *drop-out temperature*. The company can produce slabs 10 in. thick or buy 8 in. thick slabs (the length of a slab is between 5 and 10 m and the width is between 0.6 and 1.5 m). Therefore, when schedules are generated, the available slabs may be cold, warm or hot 10 in. thick slabs, or cold 8 in. slabs. The last two characteristics mentioned (thickness and charging temperature), together with the drop-out temperature determine the "processing time" of each slab in the RFs, i.e., the *residence time*.

There are two identical 45 m long RFs in the heating area, that have the capacity for reheating up to 40 slabs per furnace at the same time. The slabs in the furnaces are moved slowly from the charging door to the exit end to give to the slabs the temperature that the HSM requires to process them. Slabs are regularly removed from the exit end, shifted by one slab, and a new slab is entered into the furnace.

When a slab is charged into an RF, a residence time is calculated for that slab. The residence time determines the 'pace rate' for the slabs, which indicates how quickly the slabs should move through the RF. Every slab in the furnace has a speed assigned to it, and the speed that the company assigns to the whole furnace is calculated as the speed of the slowest slab in the furnace, i.e., slabs can be overheated. It takes 1.5–3.0 h for a slab to reach the required temperature and be discharged when the HSM "calls", i.e., when the HSM is ready for the next slab; it will trigger the release of the slab from the RF, so that there is no delay between the heating area and the HSM.

The HSM area has two sections: the *roughing mill* and the *finishing mill*. The roughing mill consists of one stand that reduces the thickness of a slab from several inches to about 30 mm. The resultant strip is then sent to the finishing mill, where there are several rolling stands to progressively reduce the thickness of the steel strip to a required final gauge (from 1.39 to 6.19 mm). The thin strip, thousands of feet in length, goes to the coiler to form the final product, the steel coil.

Since the rollers at the roughing mill and finishing mill suffer wear and tear they have to be replaced from time to time. The set of slabs produced between two consecutive changes of finishing mill rollers is called a *Product Block* (PB) and the set of slabs processed between two consecutive changes of roughing mill rollers is called a *Line-up* (LU). On average the finishing mill rolls are replaced every 8 h and the roughing mill rolls are changed every 24 h, so an LU normally has three PBs.

There are several factors that restrict the scheduling of the production of coils: product quality specifications, process efficiency standards, and target delivery due dates. Each slab has several important characteristics: width, thickness, grade (chemical composition), charging temperature, drop-out temperature, hardness, and gauge (required thickness of the coil that will be produced). The most important restrictions require smooth changes in four aspects: width, hardness, gauge, and residence time in the furnace.

Smooth changes in width of the slabs is translated into the condition that we should schedule the slabs in such a way that the width profile of a PB has a "coffin shape" (see Fig. 2). There are two parts in a PB: the wide-out, in which we schedule the slabs from narrow to wide, to warm up the rolls, and the coming-down, in which we schedule the slabs from wide to narrow, to avoid marks in the coils. The coffin shape should be started right after a change of rolls at the finishing mill. There are more complex coffins, but the one presented in Fig. 2 is the one used at Dofasco.

To warm up the rolls (break-in part), we should use 2–6 special slabs that are narrow and easy to roll. The wide-out is continued with 5–15 progressively wider pieces that are not "too hard". During the coming-down part, after a certain number of slabs of the same width are processed, the slabs mark the rolls with their edges. If after the rolls get those marks we schedule wider slabs, the marks on the rollers are transferred back to the coils and those represent poor quality products. Therefore, the slabs should be scheduled in non-increasing order of width.

Smooth changes in residence time refer to a good use of energy in the RFs. If a few slabs that require 3 h residence time are in a furnace, all the slabs in the furnace will move at the speed of those slabs; and therefore they will remain in the furnace longer than required and they will be overheated. This is a waste of energy and efficiency. For that reason changes in residence time from slab to slab should be gradual. Even more, the residence time of each slab may affect the heating time of several others before and after that slab. Therefore, it is important to have gradual changes between a slab and a group of slabs.

Smooth changes in hardness and gauge are necessary because the roughing mill and the finishing mill have to make some adjustments in the force applied to each slab. The operators prefer not to do large changes from slab to slab; therefore, those changes should be gradual.

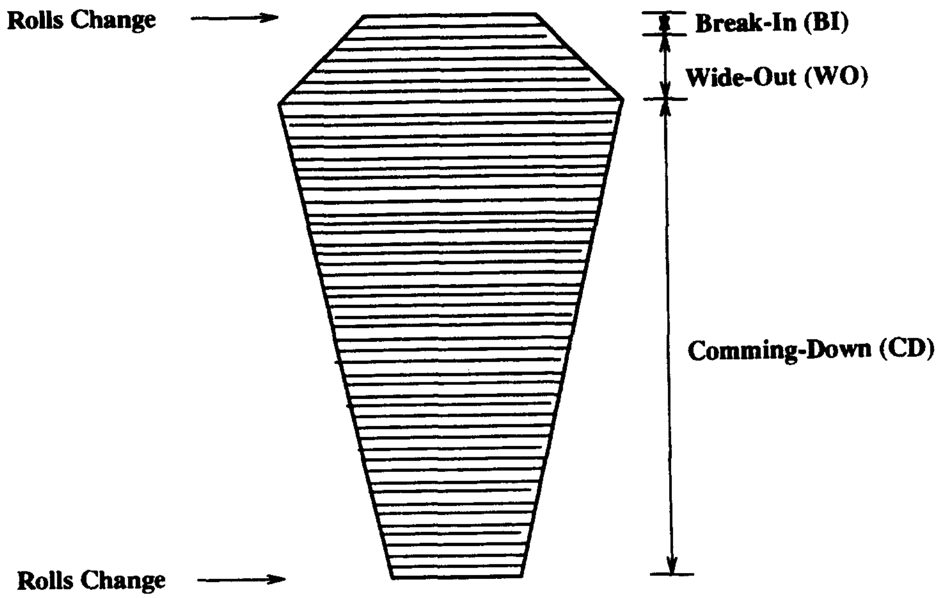

Fig. 2. Coffin shape width profile of a PB.

There are some restrictions that refer to the position that some types of slabs should have within a PB. These are called the critical products. For example, some orders that require high quality surface coils have to be processed within the first 50 slabs in a PB. If we do not schedule those slabs within the first 50 pieces, the rolls at the finishing mill would be already used and likely worn and their surface may not produce coils of high quality.

In HSM production, roll performance plays a significant role in determining product quality, operating costs, and productive capacity. If a PB has too many slabs, the rolls will wear and the worn rolls will cause poor quality steel coils. If a product block is too short, the rolls will be changed before it is necessary, which creates more frequent setups and higher cost of roll maintenance. The length of a product block depends on the roll wear, which is a function of hardness and gauge of the slabs in the PB. In the absence of a good rule, the experts at Dofasco use the guideline that a PB should be between 180 and 220 pieces. In a future version of the algorithm we will try to develop a model that uses a function of hardness and gauge to get a more accurate estimate of feasible PB length.

There are several conflicting objectives in a steel plant that are involved in the HSMPSP. Two obvious conflicting objectives are product quality and productivity. Improving product quality requires changing the rolls as frequently as possible, whereas increasing production rate requires minimizing the number of roll changes.

In order to maximize productivity, we would like to create nice homogeneous PBs (similar residence time, hardness, gauge, grade, and smooth width changes). In particular, for less common types of slabs, it is "better" to hold them until sufficient quantities are required to schedule an "efficient" PB. Sales would like orders to be produced at (or before) the due date. Producing large batches early leads to high inventories, another source of inefficiency. These three objectives, homogeneous batches, on-time delivery and low inventory levels are in conflict.

So, the problem consists of scheduling the production of the HSM, taking into account the mentioned restrictions, and trying to meet four objectives: maximize product quality, maximize productivity, maximize on-time delivery and minimize waste of energy in the RFs. There are some tradeoffs between these objectives and they will be represented in the objective function in the form of penalties. The function includes penalties for changes in five important elements: width, hardness, gauge, residence time, and a penalty for not including priority orders.

3. Literature review

There have been some attempts to solve the different problems that arise in the steel industry and the techniques used are quite varied. We will present in this section a few that are the most related to the HSMPSP.

Wright and Houck (1985) present a heuristic procedure to generate production schedules for the HSM in the steel industry. They generate an objective function, which represents the economic efficiency, in terms of three conflicting elements: product quality, production rate, and product handling (on-time delivery of finished products). That representation is given in terms of a penalty function that includes several parameters like changes in width, gauge, and hardness. Their heuristic algorithm, called UMPIRE (for Using Minimum Penalties to Improve Rolling Efficiency), modifies a given initial solution by moving slabs from their initial position to another position that improves the penalty function. They do this by using a trial-and-error process until no further improvement can be made. The improvement limit is fixed by the user. The method that they use to improve the schedules looks like "hill climbing" and therefore it will likely finish up in a local minimum. They do not present any mathematical model, do not report computational results, and do not discuss CPU time for the execution of the algorithm. In their approach they do not mention the RF constraints, and it looks like they do not consider them. Priority orders are part of the product handling element that they consider. The differences between our problem and this one is the RF area and critical orders (high quality), because they do not mention any of these aspects.

Balas (1989) presents a generalization of the TSP that can be stated as follows. A salesman who travels between pairs of cities at a certain cost, obtains a prize for each city visited and pays a penalty for each city not visited, wants to find a tour that minimizes his travel cost and penalties, while including in his tour enough cities to get a predetermined prize. He calls this problem the PCTSP. He distinguishes two important tasks: choosing slabs assigned to orders from an inventory (selection), and determining a sequence for processing the orders (sequencing). The two tasks must be solved together. He formulates the PCTSP as a TSP (sequencing part) with a knapsack-type constraint (selection part). This is a theoretical paper and the author focuses his attention on the study of structural properties of the PCTSP polytope, the convex hull of solutions to the PCTSP, and therefore does not report any real application of his solution method or talk about computational results. However, he mentions that this model is the basis for an approach that was implemented in 1985–1986 by Balas and Martin into a software package for scheduling the daily operation of a steel rolling mill. In the package they use several heuristics to find near-optimal solutions to this problem. Priority orders are considered by using node penalties. The model does not consider the RF area and some other restrictions such as critical products.

Kosiba et al. (1992) focus their research on the development of an analytical procedure for production scheduling of the HSM of a modern steel mill. They explicitly consider conflicting production objectives: product quality, production rate, and profits. They claim that combining these goals is equivalent to considering the single objective of minimizing the equipment wear. They measure the wear of rolls with a penalty structure function that reflects penalties for big jumps in three characteristics: width, gauge and hardness. The schedule with the lowest penalty will result in the lowest damage to the rolls, and fewer damaged rolls leads to higher product quality and a better production rate; hence better profits. The resulting problem is formulated and solved as an asymmetric TSP where the objective is minimize the total penalty. They employed the algorithm of Miller and Pekney (1990). To compare their solutions they obtained four sample HSM schedules, generated a first set of solutions according to "traditional scheduling procedures" at a plant. They used the UMPIRE method of Wright and Houck (1985) to find a second set of solutions for the same sample HSM schedules. A third set of solutions was obtained with their method and compared against the other two sets of solutions. The results they report refer to the penalty values. Their model performs well in comparison to the other two options; the penalty function is reduced by 44% in the worst case and by 78% in the best case with respect to the costs obtained with the other two methods. It is important to mention that they do not perform the selection task described by Balas (1989) in their model; they assume that all the existing orders will be processed and therefore they do not have to worry about selecting the orders to be satisfied first. For that reason the simple TSP is useful for their problem. However, when we choose a number of orders to satisfy from the total number of existing orders, there will remain unsatisfied orders, which is the case in our HSMPSP. In the last case the simple representation of the problem as a TSP is not accurate. The authors do not mention the RF area, the restrictions of priority and critical slabs, and delivery, in their problem.

Cowling (1995) describes a problem of generating a production plan for a steel hot rolling mill. The slabs are transferred from a slabyard to one furnace, to be reheated to a required temperature before being processed at the hot mill to produce steel coils. The slabyard of the plant has hundreds of piles that consist of up to 20 slabs. When a slab is required to be sent to the RF, an unpiling process may take place to reach the desired slab and that process may be costly and limit the production process. He models the slabyard as a digraph $G=(N,A)$, with weights on both nodes and arcs. Each node $i \in N$ represents a slab and its weight measures the desirability of processing that slab. An arc (i,j) represents the fact that slab j is scheduled immediately after slab i and its weight measures the desirability of scheduling slab j immediately after slab i. Rolling is organized into shifts, each of about 8 h. The sequence of slabs to be rolled in a particular shift is a "programme", whose width profile has a coffin shape. Each programme sequence may be considered as a "route" through the slabyard, with each slab having a "prize" associated with rolling it; then the problem can be seen as a *Prize Collecting Vehicle Routing Problem* (PCVRP). He proposes a heuristic method based on local search and Tabu Search (TS) for solving the problem. The local search method is used to find an initial sequence and the TS technique to improve that sequence. He also gives a comparison of the performance of his method and the existing manual systems using real data. He reports that his heuristic usually finds, in a matter of minutes, better solutions than the manually generated ones, and that in a few hours smaller improvements are made. He describes his results using three criteria: actual length of a solution, total arc weight, and time to generate the solutions. In all these aspects the heuristic proposed gives better results than the traditional system. The author mentions that he does not consider the problem of RFs; that all slabs are cold charged. He concentrates on the unpiling process, which is important in his problem. Even though he uses a TS technique to solve his problem, the mentioned differences make his problem different to ours.

All of the papers described in this section deal in some way with a scheduling problem at the HSM area, but none of them deal with the same problem we are facing. For this reason, we build a mathematical model that represents the HSMPSP, that includes all the restrictions mentioned in Section 2. We develop a heuristic method to approach the solution, based on the TS technique. We take some ideas expressed by some of the authors to evaluate the results that our heuristic produces.

4. Problem formulation

In this section we represent the HSMPSP as a mathematical programming problem. From this point of view, we can see that there are two important activities associated with the HSMPSP. The first one is concerned with the orders we should choose to schedule (selection), and the second one with the sequence we should assign to the selected orders (sequencing). The selection of the orders can be represented as a knapsack problem and the determination of the sequence for the selected orders can be represented as the well known (asymmetric) TSP. These two tasks have to be considered simultaneously.

The way we formulated the HSMPSP is a combination of a mathematical model and a set of constraints that we control in a computer program. The mathematical model is a generalization of the PCTSP developed by Balas (1989), which is itself a generalization of the TSP. The set of restrictions refer to quality of the surface of some coils, special slabs for the wide-out, and the RF constraints.

The PCTSP is the traditional TSP problem with a knapsack-type constraint. We can view this problem as a salesman who travels between pairs of cities i and j at a cost c_{ij}, gets a prize w_i in every city i that he visits and pays a penalty c_{ii} for each city he does not visit. He wants to minimize his travel cost and penalties while visiting enough cities to obtain a predetermined prize w_0.

Let us define the variables and parameters in our mathematical formulation: consider a graph $G = (N, A \cup O)$ where N is the set of nodes, A is the set of arcs, O is the set of loops, X is the incidence matrix of loops and arcs of $G, X \in \{0,1\}^{n^2}$; $X_{ij} = 1$ means that slab j is scheduled right after slab i, with a penalty $c_{ij}, x_{ii} = 1$ means slab i is not included in the schedule and therefore a penalty c_{ii} should be charged in the objective function. The objective function is to minimize the total penalty for scheduling slabs in a particular sequence. If we define w_i as the length (or weight) of slab i (prize in the PCTSP), $\sum_{i\in N} w_i$ is the total "available" length (total available prize in the PCTSP), since $x_{ii} = 1$ means that slab i is not scheduled; therefore the length (prize) w_i is not rolled (obtained). On the other hand, $x_{ii} = 0$ means slab i is scheduled, and the length w_i is rolled. Therefore $\sum_{i\in N} w_i x_{ii}$ represents the length not rolled (prize not obtained), and therefore $\sum_{i\in N} w_i - \sum_{i\in N} w_i x_{ii}$ is the length rolled (prize obtained). Lower and upper limits on the length that can be rolled before a roll change, can be implemented in the form: $w_0 \leqslant \sum_{i\in N} w_i - \sum_{i\in N} w_i x_{ii} \leqslant w_1$ which means that the total length rolled should be between two limits. Setting $L = \sum_{i\in N} w_i - w_1$ and $U = \sum_{i\in N} w_i - w_0$ we can re-write the restriction as $L \leqslant \sum_{i\in N} w_i x_{ii} \leqslant U$.

In addition, the PCTSP finds a tour, whereas the HSMPSP finds a sequence. Therefore, we need to add a dummy slab '0' with no sequence cost and a width of zero. We also modify the model to force slab '0' to be included. We can model the HSMPSP as follows:

$$\min \quad \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij}, \tag{1}$$

$$\sum_{j=0}^{n} x_{ij} = 1, \quad i = 0,\ldots,n, \tag{2}$$

$$\sum_{i=0}^{n} x_{ij} = 1, \quad j = 0,\ldots,n, \tag{3}$$

$$L \leqslant \sum_{i=0}^{n} w_i x_{ii} \leqslant U, \tag{4}$$

$$x_{00} = 0, \tag{5}$$

$$x_{ij} \in \{0,1\}, \quad i,j = 0,\ldots,n, \tag{6}$$

$G(x)$ has exactly one cycle of length $\geqslant 2$, (7)

Surface quality restrictions, (8)

Special pieces for the wide-out restrictions, (9)

RF restrictions. (10)

Restriction (4) can also be interpreted as a limitation on the number of slabs in a product block or line-up. Restriction (5) means we have to include the dummy slab '0' in the optimal solution. Restrictions (8)–(10) cannot easily be written in mathematical form. Restrictions (8) and (9) include "if, then, else" type of questions. Restriction (10) will be simulated.

An example solution to this model is given in Fig. 3. Cities 2, 3, 5, and 6 are included in the optimal schedule (tour) but not the others.

Let us simplify the HSMPSP, forgetting for a moment, about the surface restrictions, special slabs for the wide-out, and RFs (restrictions 8–10). The resultant model is the PCTSP. Balas and Martin (ROLL-AROUND software) find near-optimal solutions to the PCTSP, using a combination of several heuristics. It is important to mention that one of the restrictions we have relaxed, in this simplified version of the HSMPSP, is the one that refers to the RFs and this is a major relaxation since the RF area is a bottleneck in our process.

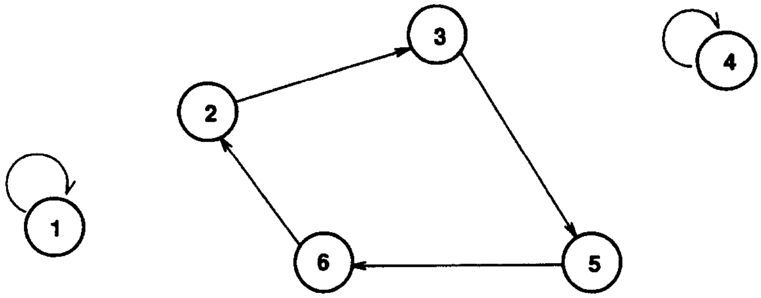

Fig. 3. An example solution for the HSMPSP.

5. The Tabu search approach

In this section we describe our approach to the HSMPSP. Since this problem is hard to solve, we elected to use a heuristic. Our heuristic is based on the TS technique and before we explain it we will briefly describe TS.

5.1. The TS technique

TS is an optimization metaheuristic independently developed by Glover (1986) and by Hansen (1986), that can be superimposed over another algorithm to guide the search to a solution. However, as in any other heuristic method, there is no guarantee that the solution is the global optimum. TS has been widely and successfully used to solve a great variety of problems, particularly difficult scheduling problems (see Adenso-Diaz, 1992; Costa, 1993; Hertz, 1991; Icmeli and Erenguc, 1994; Skorin-Kapov and Vakharia, 1993; Sun et al., 1995; Widmer and Hertz, 1989; Widmer, 1991; Wright, 1994; Cowling, 1995, and some others in Glover et al., 1993).

TS begins the search with an arbitrary initial solution (generally, the better the initial solution, the faster the method works). From the solution in one iteration, the search process evaluates "adjacent solutions" that can be reached from the current one by a simple transformation. The set of adjacent solutions is called the *neighborhood* of the current solution and it can be generated by applying a transformation to the current solution. The transformation of a current solution that leads us to a new adjacent solution is called a *move*. The transformation is characterized by a well defined rule, such as the exchange of two jobs in a scheduling problem, or the deletion of one of the jobs, or the insertion of another job. If the current solution is s and we apply the move m to it, we generate another solution s', then we can define the neighborhood as follows:

$$N(s) = \{s' | s' = s \oplus m\}, \quad s, s' \in S,$$

where S is the solution space.

TS analyzes each of the solutions in the neighborhood and takes the best of them to move the current solution to another point in the solution space. The objective function value of the new solution may either improve or deteriorate. Unlike "hill-climbing" methods, TS accepts the new solution even if it is worse than the previous one; it is trying to avoid being trapped at a local optimum and hopefully find a global optimum.

To avoid cycling during the search process, TS forbids transformations that lead the process to recent previous solutions. A move that leads the search to a previous state is called a *reverse move* and is considered *tabu* for a certain number of iterations. To control reverse moves TS uses a *tabu list* to remember the recent trajectory of the search.

To decide when to stop the search process, we can use one of several different *stopping criteria*. Some of these may be: a certain number of iterations, a predetermined amount of CPU time, a good bound (lower or upper), or a certain number of iterations without any improvement in the objective function value. Whenever one or more of these criteria are met the process is stopped and the best solution found so far is reported.

5.2. The heuristic approach

The heuristic we have developed is an aggressive method to find good solutions to the HSMPSP. It repeatedly applies the TS technique and a new idea called "*Cannibalization*". The simple TS stages improve the solutions based on exchanges of individual slabs, while the cannibalization stage consists of an improvement phase based on exchanges of groups of slabs. The cannibalization stage makes use of a method called the CROSS exchange, which has been proposed by Taillard et al. (1995) in the context of a TS heuristic for the Vehicle Routing Problem with Time Windows (see also Badeau et al., 1995). The parameters used in the application of the simple TS part are described in Section 5.4, while the CROSS exchange, the cannibalization method, and the description of our algorithm are presented in Section 5.5.

5.3. Initial solutions

A solution for the HSMPSP is a Line-up (LU), which is a set of three PBs, that satisfy some or all the restrictions on smooth changes (width, hardness, grade, and gauge), and the restrictions related to surface quality, special pieces for the wide-out, and RFs. If a slab violates one or more of those restrictions, there will be a positive penalty associated with that slab. We generate several initial solutions to begin the search process. We have developed six greedy heuristics to generate the initial solutions and they are all similar. The difference between them is the way we sort the data file and the way we choose the next slab for a PB. To start the generation of an initial solution we have a set of slabs to choose from, the slabs that are available to be scheduled right away. Each of these heuristics generate an initial solution from the same data set. Those slabs are sorted either by width within hardness groups (type A groups) or just width (type B group). The first slab in the coming-down part is the widest available. We have different ways to choose the next slab given the current slab i:

- The next available slab j in one particular hardness group (one specific group of type A) that has a penalty c_{ij} less than or equal to a predetermined value.
- The next available slab j in any hardness group (any group of type A) that has a penalty c_{ij} less than or equal to a predetermined value.
- The next available slab j in the group sorted only by width (the unique group type B) that has a penalty c_{ij} less than or equal to a predetermined value.
- The next available slab j in one particular hardness group (one specific group of type A) that has a minimum penalty c_{ij} (it may be less or greater than the predetermined penalty of the first three methods).
- The next available slab j in any hardness group (any group of type A) that has a minimum penalty c_{ij}.
- The next available slab j in the group sorted only by width (the unique group type B) that has a minimum penalty c_{ij}.

The algorithms that generate the initial solutions are "greedy" in the sense that they always choose the best available slab, with no backtracking. To generate those solutions we choose the best slab from the corresponding data base, according to one of the selection methods described, and incorporate it into the current PB until either we have an adequate number of slabs in the PB or we cannot find any more feasible slabs.

Some of the methods used report better solutions than the others, but all of them are useful to choose the initial solutions that will be improved with the application of our TS method.

5.4. Parameters used in the basic TS algorithm

Since the TS technique is used in stages one and three of this heuristic approach, we will briefly describe how this technique is tailored to this application. The simple version of the TS technique used in this paper follows the general guidelines provided by Glover (1990, 1991). After we find the initial solutions we have two sets of slabs: the scheduled, and the unscheduled that are available to be scheduled at any time. To improve an initial solution we exchange scheduled and unscheduled slabs in a systematic way until we reach a stopping criterion.

We move from a current solution to another solution in the neighborhood.

Move: To select a relatively small subset of moves to expedite the process, our *move* is based on swaps of a single scheduled slab and a single unscheduled one, that is, find a slab j (scheduled between slabs i and k) with positive penalty p_{ij} and look for all unscheduled slabs r such that

$\text{Width}_i \leqslant \text{Width}_r \leqslant \text{Width}_k$

for pieces in the wide-out part,

or $\text{Width}_i \geqslant \text{Width}_r \geqslant \text{Width}_k$

for pieces in the coming-down part,

and select the best of them, based on minimizing the penalty $p_{ir} + p_{rk} - p_{ij} - p_{jk}$. The swap may be considered as the insertion of slab r and the deletion of slab j.

Neighborhood ($N(s)$): The neighborhood of a current solution s is defined as the set of all schedules s' that can be obtained by the application of the move.

Tabu list: Each time we perform a swap, we keep in memory the deleted and inserted slabs and keep the reverse moves tabu for a predetermined number θ of iterations, the size of the tabu list, which in this application is set to 10. If we deleted slab j in an iteration, the insertion of that slab is a reverse move and it will be forbidden for θ iterations. On the other hand, if in an iteration we inserted slab r, the deletion of that slab is a reverse move and it will be forbidden for θ iterations. Then, the tabu list includes the last θ reverse moves that are tabu. This tabu list is updated at each iteration.

Stopping criteria: We have chosen two common stopping criteria to terminate the search process. They consist of a fixed number of iterations (50) and a fixed number of iterations without improvement on the value of the objective function (20).

5.5. The algorithm

The methodology used in this research to solve the HSMPSP makes use of a recent method that generalized previous edge exchange heuristics for the Vehicle Routing Problem with Time Windows. This method is called the CROSS exchange, and its adaptation to the HSMPSP is illustrated in Fig. 4. Let us assume we have two product blocks. Before performing the CROSS exchange, slab i'_1 is scheduled right after slab i_1, and slab j'_1 is scheduled right after slab j_1 in PB # 1. Similarly, slab i'_2 is scheduled right after slab i_2, and slab j'_2 is scheduled right after slab j_2 in PB # 2. The CROSS exchange consists of swapping two blocks of slabs defined by the sets $\{i'_1, \ldots, j_1\}$ and $\{i'_2, \ldots, j_2\}$. After the swap, slab i'_2 is scheduled right after slab i_1, and slab j'_1 is scheduled right after slab j_2 in the new PB # 1. Similarly, slab i'_1 is scheduled right after slab i_2, and slab j'_2 is scheduled right after slab j_1 in the new PB # 2.

The *cannibalization method* that we propose consists of applying CROSS exchange between *bad sections* of a PB, that is sections that contain slabs with positive penalty, and *good sections* of other PBs, that is, sections with lower penalty or sections that reduce the penalty of the first PB. The objective of the cannibalization method is to improve a current solution (LU) by improving each of its PBs by eliminating their bad sections and taking good sections of other PBs that belong to the other solutions (LUs).

The way we determine a bad section is as follows: we inspect a PB from the beginning until we find a slab with positive penalty (a *bad slab*), which determines the beginning of the bad section. Then we continue moving to the next slabs in the PB, asking if the next slab is bad or not. If that slab is bad we include it in the bad section, otherwise we inspect the next slab. We repeat this process until we do not find a bad slab in the next 10 consecutive inspected slabs. The last bad slab is the ending slab of the bad section. In this way we know the first (*first bad*) and the last (*last bad*) slabs in the bad section, and therefore we know their characteristics and the number of slabs in the bad section. We also know the slab previous to the starting slab (the *previous* slab) and the one after the ending slab (the *next* slab), and therefore their width.

The way we determine a good section in a PB is as follows: since we know how many slabs there are in the bad section and the width of *previous* and *next* slabs, we can look for groups of slabs

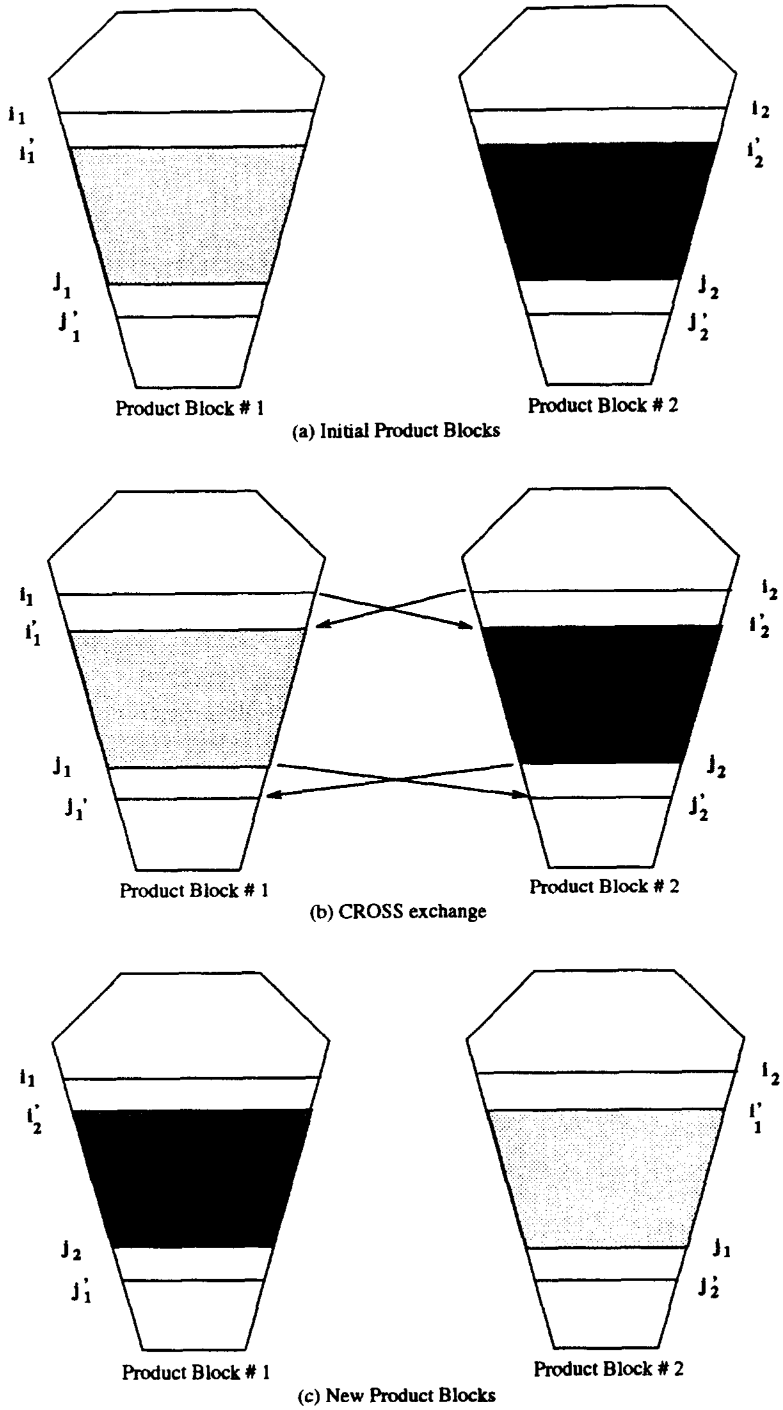

Fig. 4. The CROSS exchange for the HSMPSP.

of similar width to *previous* and *next* (the *first good* slab in the good section should be a little narrower that *previous* and the *last good* slab in the good section should be a little wider than *next*). If the accumulated penalty of the slabs in that group is less than the accumulated penalty of the bad section we declare that group of slabs as a good section. If we find more than one good section, we will choose the one that produces the maximum reduction of the LU penalty. That good section will produce the best CROSS exchange.

The penalty of a section of a PB is the sum of the penalties of the slabs included in that section. Fig. 5 illustrates the cannibalization heuristic method to improve a PB. In the figure, PB # 1 is improved by exchanging its four bad sections by

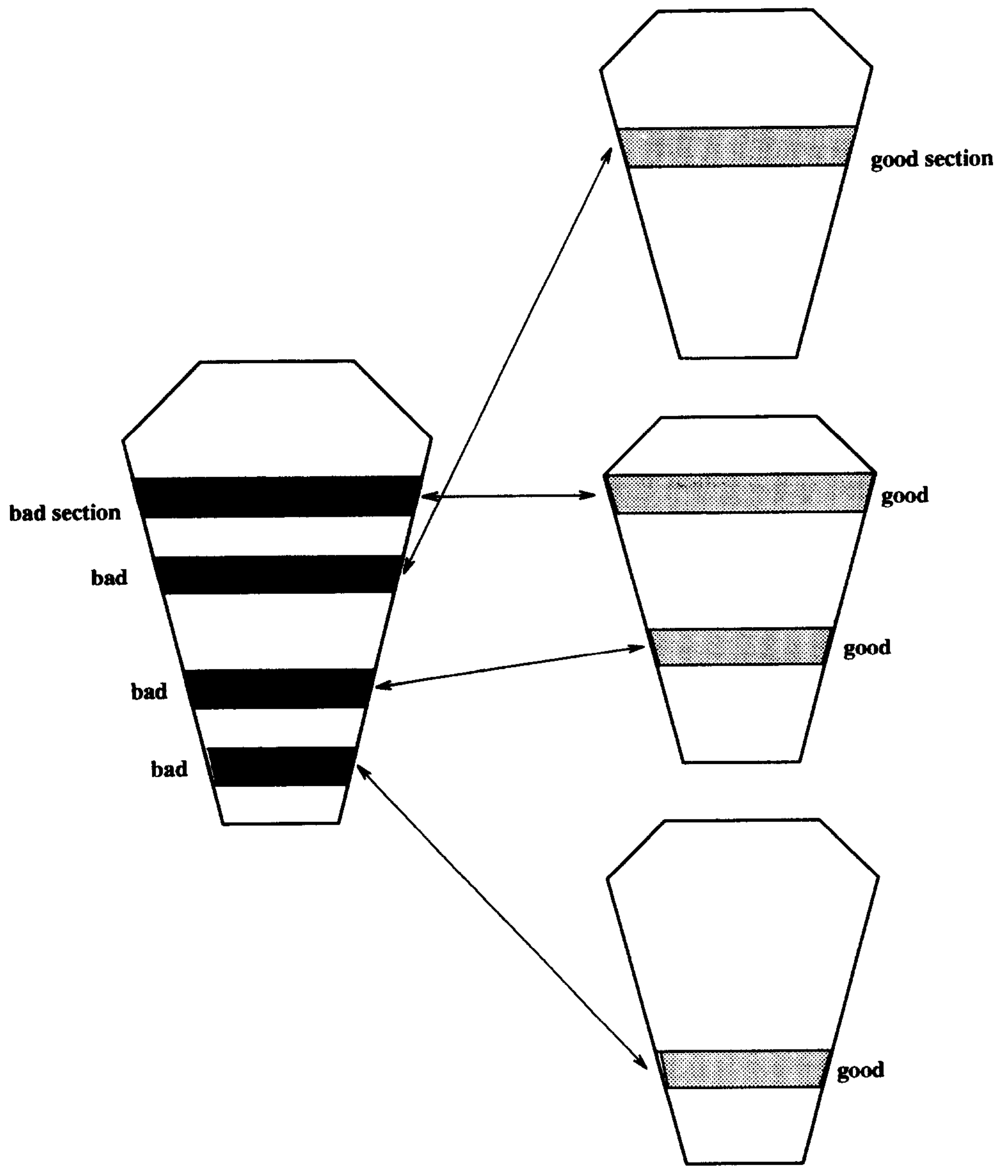

Fig. 5. The cannibalization method.

good sections from the other three PBs. For simplicity, each CROSS exchange in Fig. 5 is represented by a "two way" arrow instead of the four "one way" arrows used in Fig. 4.

Our heuristic begins by generating K initial solutions (K "disjoint" LUs); then it iteratively improves them in three stages: The first one is a false start that applies a "short" TS (five iterations of a simple TS) on each initial solution to ensure that we start with a reasonable solution (Step 1 in the algorithm given later). The second stage improves the current solution based on swaps of groups of slabs from different solutions. This stage is the cannibalization technique, which exchanges the bad sections of the PBs of the current solution (LU) with good sections of the PBs of the other $K-1$ solutions (LUs) to try to improve the current solution (Step 2). A CROSS exchange is applied at this stage to swap bad sections of the current solution and good sections of the other solutions. The third stage (Step 3 of the algorithm) applies a long TS (50 iterations) to improve the current solution based on the individual swaps (moves) defined in Section 5.4. We repeat stages two and three until the stopping criterion is met. The best solution found is the one we report.

This heuristic requires a tabu list for the simple TS (as defined in the previous section) and a *cannibal list* for the cannibalization method, which is a tabu list for the cannibalization stage. The cannibal list is made up of good and bad sections that were exchanged in recent iterations. If a bad section in iteration k (solution k) from product block i is swapped with a good section from solution l, product block j, that is, a "section (k,i)" was deleted from solution k (and inserted in solution l), and a "section (l,j)" was inserted in solution k (and deleted from solution l), in the next five iterations we cannot swap any bad section from solution l, PB j, and any good section from solution k, PB i because we could be trapped in a cycle. However any bad section (l,j) can be interchanged with any other good section from other product blocks. This heuristic also needs two stopping criteria, one for the simple TS (defined in Section 5.4) and the other for the cannibalization part. The cannibalization stopping criterion is that if the number of LUs examined with no improvement of the current best solution is equal to the number of initial solutions, K, stop; the algorithm has already examined all of the existing solutions and it did not find any improvement of the current best solution. The algorithm is as follows:

Step 0 (*Initialization*).
Generate K initial solutions (line-ups) in the following way:
(i) From the set of available slabs generate m trial solutions (using m simple heuristics)
(ii) Select the best of them and eliminate the others. Delete the selected slabs from the set of available slabs
(iii) Repeat steps (i) and (ii) K times
Set $k := 1$,
number of line-ups examined without improvement := 0,
best_found_solution := kth line-up.

Step 1 (*False Start*).
Perform a short TS on each initial solution.

Step 2 (*Cannibalization Phase*).
For each bad section of the kth line-up, repeat:
- Select the best CROSS exchange for the current bad section among all the other line-ups.
- Perform this exchange if this improves the value of the current kth solution. [1]
- Update the cannibal list.

Step 3 (*Tabu Search*).
Perform a long tabu search to line-up k.

Step 4. If the best solution has improved during Steps 2 and 3,
- then set the number of line-ups examined without improvement equal to zero and best_found_solution := kth lineup,
- else increase the counter of the number of line-ups examined without improvement,

set $k = k + 1$.

[1] It is possible that a bad section from kth LU might improve one of the other LUs, but we do not consider it explicitly in the algorithm. The procedure will cycle through the other LUs, trying to improve them and will discover the improvement in a later iteration.

If $k > K$, then set $k := 1$,

Step 5 (*Stopping Criterion*).

If the number of LUs examined without improvement is equal to K,

– then stop, report the best found solution,

– else, go to Step 2.

6. Computational results

This algorithm was programmed in C++ and runs on a personal computer. We have tested our heuristic on several production data files from Dofasco in Hamilton, Ontario, Canada, and we generated 18 PBs. A summary of statistics relative to four important parameters is reported in Table 1. The parameters measured are the number of slabs in a PB, the number of priority orders included in the PB, the percentage of priority orders included in the PB, and the PB penalty. The values of the parameters described in the previous section and used in our heuristic are as follows:

- Number of trial solutions, $m = 6$, (using the six greedy heuristics described in Section 5.3),
- Number of initial solutions, $K = 4$,
- Size of the cannibal list = 5.

The structure of the penalty function was discussed with the experts at Dofasco and they suggested the penalties given in Tables 1–3. We know that a continuous penalty scale would make more sense, but the experts do it in discrete limits.

WiDif = Width(i − 1) − Width(i)	Difference in width
HGDif = HG(i − 1) − HG(i)	Difference in hardness
GGDif = GG(i − 1) − GG(i)	Difference in gauge group
RTDif = RT(i − 1) − RT(i)	Difference in residence time
MaxRTDif	Maximum residence time difference allowed

The penalty for scheduling order j right after i is given by the sum of the four previously described penalties plus a penalty of 100 units if a scheduled order is not priority, that is

$$\text{Penalty}_{ij} = \text{WidthPen}_{ij} + \text{HGPen}_{ij} + \text{GGPen}_{ij} + \text{RTPen}_{ij} + \text{PrioPen}_{j}.$$

The results obtained with our heuristic are compared against the results obtained by the current manual "computer-assisted" method that Dofasco uses to generate LUs. We computed the same statistics from 30 actual PBs generated with the scheduling system currently in use and compared them against the results generated by our heuristic.

Table 1
Penalties due to changes in width

WidthDif	WidthPen
Break-in	
$-100 \leqslant$ WiDif $\leqslant 0$	0
<-100	1000
>0	1000
Wide-out	
<-100	1000
<-50	500
<-30	200
<-20	0
$\leqslant 0$	200
>0	1000
Coming-down	
<-50	1000
<-25	500
<0	100
$\leqslant 100$	0
$\leqslant 125$	100
$\leqslant 150$	500
>150	1000

Table 2
Penalty due to changes in hardness and gauge

Abs(HGDif)	HGPen	Abs(GGDif)	GGPen
$\leqslant 2$	0	$\leqslant 2$	0
>2	1000	>2	1000

Table 3
Penalty due to changes in residence time

Abs(RTDif)	RTPen
$\leqslant$ MaxRTDif	0
$\leqslant$ MaxRTDif + 10	200
$\leqslant$ MaxRTDif + 20	500
$>$MaxRTDif + 20	1000

Table 4 shows five columns and four sets of four rows that refer to the four parameters that we are measuring. The first column indicates the parameter that we are analyzing; the second one shows the values obtained with the current system; the third one shows the values obtained with our computer program without the use of TS, i.e., the initial solutions; the fourth one reports solutions obtained with a simple application of TS; and the last one refers to the use of the cannibalization heuristic. The first set of four rows refers to the number of slabs included in a PB; the second set shows the number of priority orders included in the PB; the third set includes the percentage of the priority orders included in a PB; and the fourth set reports values of the penalty function. The four rows in each set of rows refer to statistics observed in the PBs analyzed.

From Table 4 we can conclude that the average number of slabs included in a PB with our method is higher than the average under the current system and that number is also more consistent (smaller standard deviation and extreme values). The increment observed represents an average of 14%. Even more, the current system generates longer sequences than that suggested by the experts (they scheduled up to 325 when the maximum allowed is 220). The number, and therefore the percentage, of priority orders included in our PBs are superior to the ones observed with the current method. With respect to the value of the penalty function, the one obtained with our heuristic is consistently smaller than the values reported for the current system, even if we do not use any optimization method. The average penalty is reduced by 82% before optimization and by 89% with the cannibalization method. If we compare the two types of solutions generated with our system, we can say that the number of priority orders considered is about the same, but the inclusion of the cannibalization method reduces the PB average penalty of the initial solutions by 39%.

To compare the same results from a different perspective, we report the three best PBs generated by the current system and the three best PBs generated with our heuristic. That selection was made based on three parameters: the number of slabs in a PB, the percentage of priority orders included in the PB, and PB penalty. According to the people that verify and authorize the actual production of the schedules generated, a good number of slabs in a PB is between 180 and 220. The results of the best PBs are reported in Table 5, where the penalty is represented in thousands. We observe that the

Table 4
Summary of results of the current scheduling system and the proposed heuristic

Parameters	Current system	Initial solution	Cannibalization
Min # of slabs/PB	96	214	214
Max # of slabs/PB	325	215	215
Average # of slabs/PB	189	215	215
Std. dev. # of slabs/PB	45	0.5	0.5
Min # priority slabs/PB	11	162	162
Max # priority slabs/PB	128	210	204
Avg # priority slabs/PB	50	199	195
Std. dev. prio. slabs/PB	32	13	12
Min % priority slabs/PB	4	76	76
Max % priority slabs/PB	64	98	95
Avg % priority slabs/PB	27	93	91
Std. dev. % prio. slabs/PB	16	6	5
Min penalty/PB	13 000	2800	1500
Max penalty/PB	97 200	15 000	11 450
Average penalty/PB	45 533	8041	4941
Std. dev. penalty/PB	20 621	4502	3151

Table 5
Comparison of the best three PBs

Best PB #	Current system			Initial solution			Cannibalization		
	1	2	3	1	2	3	1	2	3
# Slabs	182	212	271	214	215	215	214	215	215
# Priority slab	107	128	56	204	205	210	204	198	202
% Priority slab	59	60	21	95	95	98	95	92	94
Penalty	18.4	31.0	34.9	3.2	2.8	4.65	1.5	2.45	1.85
Average penalty	28 100			3550			1933		

number of priority orders included in the actual PBs varies between 21% and 60%, whereas in our method this percentage varies between 92% and 98%. With respect to penalty values, the actual PBs vary between 18 400 and 34 900 units, whereas the ones obtained with our method vary between 2800 and 4650 units before the TS application and between 1500 and 2450 with the cannibalization method. These numbers imply a reduction in the penalty function equivalent to 87% before our optimization method is used and 93% after cannibalization.

Finally, with respect to the time required to generate each PB, we can say that, on average, a good initial PB is generated in less than 5 min before any optimization; in about 3–5 more minutes we obtain a very good solution with the application of a basic TS algorithm; and in about four more minutes we obtain the final result with the cannibalization method. Therefore, we can say that, on average, in less than 15 min we can generate a very good PB. The current system used in the company generates a PB in about 2 h.

7. Conclusions

In this paper we have described the HSMPSP that arises in a steel company and formulated it as a mathematical programming model. We have proposed a heuristic method that uses the TS metaheuristic that gives very good solutions to this hard problem.

From the results shown in Tables 4 and 5, we can conclude that our heuristic represents a considerable improvement over the current approach in several aspects: size of PBs, percentage of priority slabs included, penalty values, and the time to generate PBs. These improvements should have some positive consequences for the company.

First, it increases the size of the PBs, keeping the size of each PB always in the recommended range. Long PBs without sacrificing the quality of the products, imply longer use of rolls and less frequent changes of them. This results in two benefits: lengthening of roll life due to less grinding and reduction in the scheduled delays for roll changing. This translates into reductions in the cost of replacing and grinding rolls, better productivity rate, and likely better use of resources in the RFs due to less overheated slabs. This will represent some cost savings for Dofasco.

Second, it generates schedules that include a much higher percentage of priority orders, that is, orders that are either urgent, near their due date, or already old. This implies better delivery times and therefore better relationships with customers.

Third, the penalty function has been reduced dramatically, which means that there are only a few violations of the restrictions, both at the RFs and at the hot mill. This property implies a reduction in set-up times in the hot mill and therefore possible reduction of delays of slabs in the RFs and better use of energy in the furnaces, which are frequently the "bottleneck" of the primary operations in the plant. Low penalties may also represent better use of facilities at the hot mill, and increase in quality of products.

Fourth, our heuristic can generate PBs much faster than the traditional method. The time to generate PBs is reduced from 2 h to 15 min.

We do not claim that the heuristic method finds the optimal solution to the HSMPSP. The difficulty of the problem and the tradeoffs that exist between each of the conflicting objectives mentioned make it computationally intractable. However our heuristic produces excellent solutions that have out-performed the current system to generate schedules for the HSM. The company is very excited about these results and plans to do parallel testing in the next few months. We plan to schedule the production for three days, using the same data base from real production data that they will use, and repeat this for three consecutive days. In that way we will have nine LUs i.e. 27 PBs to compare against.

Acknowledgements

This research was partially supported by the Ontario–Quebec Projects of Exchange, the National Bureau of Science and Technology of Mexico (CONACYT), the Natural Sciences and Engineering Research Council of Canada (NSERC), the University of Las Americas Puebla (UDLAP), and Dofasco. Their support is gratefully acknowledged. We are grateful to two anonymous referees for their thorough review and many useful suggestions.

References

Adenso-Diaz, B., 1992. Restricted neighborhood in the tabu search for the flowshop problem. European Journal of Operational Research 62, 27–37.

Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, E., 1995. A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. Centre de Recherche sur les Transports, Publication CRT-95-84.

Balas, E., 1989. The prize collecting traveling salesman problem. Networks 19, 621–636.

Balas, E., Martin, ROLL-AROUND: Software package for scheduling the rounds of a rolling mill. Copyright Balas and Martin Associates, Pittsburgh, PA.

Cheng, T.C.E., 1993. Parallel flowshop scheduling with a common machine. Hong Kong Polytechnic, Working Paper.

Costa, D., 1993. An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR 33 (3), 161–178.

Cowling, P., 1995. Optimization in Industry (ch: Optimization in steel hot rolling). Wiley, NY.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13, 533–549.

Glover, F., 1990. Tabu search – Part I. ORSA Journal on Computing 1 (3), 190–206.

Glover, F., 1991. Tabu search – Part II. ORSA Journal on Computing 2 (1), 4–32.

Glover, F., Laguna, M., Taillard, E., de Werra, D. (Eds.), 1993. Tabu Search (special issue). Annals of Operations Research 41.

Gupta, J.N.D., 1988. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research 39 (4), 359–364.

Gupta, J.N.D., Tunc, E.A., 1991. Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. International Journal of Production Research 29 (7), 1489–1502.

Hansen, P., 1986. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Hertz, A., 1991. Tabu search for large scale timetabling problems. European Journal of Operational Research 54, 39–47.

Icmeli, O., Erenguc, S.S., 1994. A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows. Computers and Operations Research 21 (8), 841–853.

Kosiba, E.D., Wright, J.R., Cobbs, A.E., 1992. Discrete event sequencing as a traveling salesman problem. Computers in Industry 19, 317–327.

Lim, C.J., McMahon, G.B., 1994. The three-machine flow-shop problem with arbitrary precedence relations. European Journal of Operational Research 78, 216–223.

Miller, D.L., Pekney, J.P., 1990. Exact solution of a large asymmetric traveling salesman problem. Science 251, 754–760.

Narasimhan, S.L., Mangianelli, P.M., 1987. A comparison of sequencing rules for a two-stage hybrid flow shop. Decision Sciences 18, 250–265.

Rajendran, C., Chaudhuri, D., 1992. A multi-stage parallel-processor flowshop problem with minimum flowtime. European Journal of Operational Research 57, 111–122.

Skorin-Kapov, J., Vakharia, A.J., 1993. Scheduling a flow-line manufacturing cell: A tabu search approach. International Journal of Production Research 31 (7), 1721–1734.

Sriskandarajah, C., Ladet, P., 1986. Some no-wait shops scheduling problems: Complexity aspect. European Journal of Operational Research 24, 424–438.

Sriskandarajah, C., 1993. Performance of scheduling algorithms for no-wait flowshops with parallel machines. European Journal of Operational Research 70, 365–378.

Sun, D., Batta, R., Lin, L., 1995. Effective job shop scheduling through active chain manipulation. Computers and Operations Research 22 (2), 159–172.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., 1995. A New Neighborhood Structure for the Vehicle Routing Problem with Time Windows. Centre de Recherche sur les Transports, Publication CRT-95-66.

Widmer, M., Hertz, A., 1989. A new heuristic method for the flow shop sequencing problem. European Journal of Operational Research 41, 186–193.

Widmer, M., 1991. Job shop scheduling with tooling constraints: A tabu search approach. Journal of the Operational Research Society 42 (1), 75–82.

Wright, J.R., Houck, M.H., 1985. An application of systems modeling in steel production scheduling. Engineering Software IV, Proceedings of the Fourth International Conference, Kensington Exhibition Center, London, England, 15.127–15.140.

Wright, M., 1994. Timetabling country cricket fixtures using a form of tabu search. Journal of the Operational Research Society 47 (7), 758–770.

Yanney, J.D., Kuo, W., 1989. A practical approach to scheduling a multistage, multiprocessor flow-shop problem. International Journal of Production Research 27 (10), 1733–1742.