CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

The Cold Rolling Mill Scheduling Problem: A New Formulation and a Perturbation Solution Approach

Fayez F. Boctor

August 2016

CIRRELT-2016-42

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2016-010.

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

The Cold Rolling Mill Scheduling Problem: A New Formulation and a Perturbation Solution Approach

Fayez F. Boctor*

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Operations and Decision Systems, Université Laval, Pavillon Palasis-Prince, 2325, de la Terrasse, Québec, Canada G1V 0A6

Abstract. This paper provides a new formulation for the problem of sequencing and scheduling jobs on a single cold rolling mill. It also presents two new heuristics to solve the problem. Using three hundred instances having the same characteristics as real-life problems observed in an aluminium rolling facility; it will be shown that the proposed heuristics perform remarkably well.

Keywords. Sequencing, scheduling, heuristics, production planning, cold rolling mills.

Acknowledgements. This research work was partially supported by grant OPG0036509 from the National Science and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

---

* Corresponding author: Fayez.Boctor@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2016

1. INTRODUCTION

This paper deals with the problem of scheduling rolling jobs on a single cold rolling mill which hereafter will be called the *Cold Rolling-Mill scheduling Problem* (CRMSP). The problem studied here is a real one that has been observed in a Canadian aluminium rolling facility where the usual practice is to construct an operations schedule for the following five days on a sliding horizon basis. This plan is updated every day to reschedule the remaining tasks of the previous plan along with new ones that are planned to arrive during the coming five working days. Usually the number of jobs to be included in the plan is between 15 and 25 jobs. In this setting, the arrival date, due date, processing time, weight, final width and the final thickness of the aluminium coils to be rolled are known and deterministic. The operations' schedule should be constructed with the objective of minimizing the sum of setup costs, tardiness penalties and inventory holding costs. A setup is needed whenever the width of the coil to be rolled is larger than the width of the previous one as the edges of the rolled coil leave marks and scratches on the rollers. Thus, if the next coil has a larger width than the previous one, the rollers should be grinded and polished before rolling the larger job. The work-in-process holding cost depends on the alloy composition, the weight of the coil to be processed and its flow time in the shop (the difference between its finish and arrival dates).

In the studied industrial setting the setup cost is estimated at $300. This is the sum of the cost of replacing the working rolls by grinded and polished ones and the cost of grinding and polishing the old ones. The setup time is about 1.5 hours and the tardiness penalty is set at $30 per 1000 pounds and per hour. The average inventory holding cost per 1000 pounds and per hour is estimated at about $0.025 and the width of jobs is between 40 and 62 inches, while the job processing time is usually between 1 and 5 hours. Jobs of longer processing time or of larger or smaller widths can be also processed but are not very frequent.

2. LITERATURE REVIEW

Very few published papers deal with the rolling mill scheduling problem and all but one, address the hot mill scheduling problem. The objective in the hot mill scheduling problem is to synchronize the operations of the furnaces or the smelters with the operations of the rolling facility or alternatively, to make sure that the capacity of the rolling facility is fully used to efficiently handle the rolling jobs within a suitable window of time. This objective is usually expressed as maximizing the throughput of the facility or minimizing the make span of processing the set of rolling jobs. As underlined by Nicholls (1994), if the rolling facility is not able to handle all the required jobs in time, the entire plant can be disrupted causing substantial cost increase.

The sole paper that dealt with the cold rolling-mill scheduling problem is the paper by Mayrand *et al.* (1995) who propose a non-linear mathematical formulation of the problem and an adaptation of the genetic algorithm to solve it. The cost elements of the objective function are discounted in function of the arrival date of each job. This discount factor exponentially decreases as the arrival date increases. They do not consider due dates as either a soft or a hard constraint. Instead, they require that all jobs be finished before a chosen date. This date should be determined in a way that guarantees the feasibility of the mathematical model. However, there is no guarantee that the genetic algorithm they propose will always produce a feasible schedule even if such a schedule exists.

Several versions of the hot rolling mill scheduling problem were studied. The most studied version is the scheduling problem within a plant composed of a smelter (or melting furnace) and a rolling facility. Again, this requires that the rolling facility be able to handle all the required rolling jobs in order to avoid substantial disruptions or excessive hardening of the slabs to process. Sasidhar and Achary (1991) formulate the problem as a maximal flow problem in a multiple arc network and propose an algorithm to solve it. Nicholls (1994) proposes a mixed-integer linear program to solve this problem. Stauffer and Liebling (1997) propose a scheduling algorithm based on the Tabu search. Lopez *et al.* (1998) model the problem as a prize collecting

travelling salesman problem and propose a Tabu search heuristic to solve it. However we notice that the resulting schedule does not necessarily include all the required jobs.

Tang *et al.* (2000) model the hot rolling scheduling problem as a multiple travelling salesman problem and provide a genetic algorithm to solve it. The objective function they use is the sum of penalty cost for production changeover. The presented results show an improvement of 20% over the manual scheduling method used in practice. De Ladurantaye *et al.* (2007) revisit the problem and propose a two-phase approach to minimize a weighted function of the rolling mill idle time and a penalty of violating some constraints related to production quality. In the first phase of their approach batches of ingots to be processed are constituted and in the second phase batches are sequenced on the mill.

Chen *et al.* (2008) model the problem as a vehicle routing problem and develop a heuristic approach based on quantum particle swarm optimization and on simulated annealing to solve it**.** Also, Cowling (2003) describes a decision support system designed to help planning the operations in steel rolling facilities. This decision support system uses the Tabu search method to develop the required schedule. However, very few details are given about how this decision support system constructs the required schedules.

As'ad and Demirli (2010) deal with the problem in a process whereby only one major setup is made at the beginning of a shift. They propose a model to determine the raw material to buy and to process during this shift with the possibility of fulfilling the demand of lower grade steel (grade 40) with a higher grade (grade 60).

It is important not to assume that the cold rolling mill scheduling problem studied in this paper is a special case of what is called in the literature the "problem of scheduling a tandem of cold rolling mills" (see Dixit et al 2000, Wang et al 2000, Yang et al 2008 and Zhao et al 2008). In this case a tandem of two or more rolling mills is used to reduce the thickness of the rolled material (sheets or slabs) and the problem is to determine some of the operation parameters such as amount of thickness reduction, inter-mill tension, rolling speed and pressure for each mill with the objective of minimizing the consumed power. This tandem problem is quite different from the one studied in this paper.

Finally let us notice that although the literature studied a large collection of single machine scheduling problems (see Pinedo 2012), to the best of my knowledge, there is no publication that studied a scheduling problem that has the same constraints and objective as the cold rolling mill scheduling problem studied hereafter.

This current paper rather, deals with scheduling the execution of a set of jobs with the objective of minimizing the sum of setup, tardiness and inventory holding costs. The paper provides a new and linear formulation of the problem which considers due dates explicitly. Unfortunately, this model is difficult to solve and very time consuming. We tried to solve some test problems with only 15 jobs using the MIP code GUROBI 6.5.1 and a computer equipped with a 3.50 GHz, 8 cores Intel Xeon processor. After 10 days of computation the first problem was not solved. However, problems with 10 jobs were solved within about 62 seconds of computation time in average.

That is why we also developed two heuristic approaches to solve the CRMSP: a perturbation-based heuristic approach (PBHA) and a greedy randomized adaptive search procedure (GRASP). Within the perturbation approach an initial schedule is constructed by using a new *Greedy Multi-Rule Construction Heuristic* (GMRCH). The schedule is then improved by consecutively applying two local improvement procedures that search two different neighborhoods of the obtained solution. Then, the obtained solution is perturbed (modified) and improved using the same local improvement procedures. This is repeated a number of times.

To evaluate the performance of the developed heuristics, three sets of one hundred test-instances each are generated and used. These randomly generated test-instances have the same characteristics as the real-life problem observed in the industrial setting. These three instance sets have respectively 10, 15 and 25 jobs. As it will be shown, the proposed heuristics always produced the optimal solution for the 10-job instances. For the 15-job instances we compare the heuristic solutions to those obtained by the commercial MIP code GUROBI 6.5.1 after one hour of computation time. The cost of the heuristic solutions was, in average, 1.07 % better.

3. MATHEMATICAL FORMULATION

This section provides a new and linear formulation of the CRMSP using the following notation:

- N number of jobs to be scheduled
- j job index; j=1,..., N
- S cost of one set-up
- s time required to perform one set-up
- H a large number
- h_j unit inventory holding cost per unit of time and per unit of weight of job j
- a the earliest date where the rolling mill is available to process the considered jobs
- a_j arrival date of job j
- d_j due date of job j
- w_j total weight of job j (e.g. in pounds)
- L_j coil width of job j (e.g. in inches)
- L the largest width that can be processed at the beginning of the planning horizon without any prior set-up (L is the width of the last processed job before the beginning of the horizon)
- p_j processing time of job j (e.g. in hours)
- t_j starting date of job j
- r_j tardiness of job j; $r_j = max\ (0, t_j+p_j-d_j)$
- P_j tardiness penalty per unit of time for job j
- x_{jp} binary variable equals 1 if job j is in position p in the production sequence; p=1,.., N
- y_p binary variable equals 1 if a setup is needed before the job in position p

Due dates will be considered as soft constraints. Consequently, a penalty P_j per each tardiness time unit is added to the total cost. The value of P_j is either contractually fixed or can be set to a suitable value in order to reduce tardiness.

Using the above given notation our CRMSP can be modelled as follows:

Find: $t_j \geq \max(a, a_j)$, $r_j \geq 0, y_p \in \{0,1\}$ and $x_{jp} \in \{0,1\}; j = 1,..N, p = 1,..,N,$ which:

$$\text{Minimize: } \sum_{p=1}^{N} Sy_p + \sum_{j=1}^{N} \{h_j w_j (t_j + p_j - a_j) + P_j r_j\} \quad (1)$$

$$\text{Subject to: } \sum_{p=1}^{N} x_{jp} = 1;\ j = 1,..,N \quad (2)$$

$$\sum_{j=1}^{N} x_{jp} = 1;\ p = 1,..,N \quad (3)$$

$$t_j - sy_1 \geq H(x_{j1} - 1);\ j = 1,..,N \quad (4)$$

$$t_i - t_j - p_j - sy_p \geq H(x_{j,p-1} + x_{ip} - 2);\ j = 1,..,N, i = 1,..,N, i \neq j, p = 2,..,N \quad (5)$$

$$Hy_1 \geq H(x_{i1} - 1) + (L_i - L);\ i = 1,..,N \quad (6)$$

$$Hy_p \geq H(x_{j,p-1} + x_{ip} - 2) + (L_i - L_j); j = 1,..,N, i = 1,..,N, i \neq j, p = 2,..,N \quad (7)$$

$$r_j \geq t_j + p_j - d_j;\ j = 1,..,N \quad (8)$$

The first term of the objective function (1) is the sum of setup costs, the second term is the total inventory holding cost, and the third term gives the overall tardiness penalty. Constraints (2) state that each job has one and only one position in the execution sequence while constraints (3) make sure that there is only one job in each position. Constraints (4) set the starting time of the first job in the sequence. Constraints (5) imply that, if job *j* immediately precedes job *i*, then the starting time of *i* is greater than or equal to the finish time of *j* plus the setup time *s*, if a setup is necessary. Constraints (6) set the value of y_1 to 1 if the width of the first job is larger than *L*. Constraints (7) set the value of setup variable y_p to 1, if the job in position *p* is larger than the job in position *p*-1. Finally constraints (8) determine the tardiness r_j of each job *j*.

This model contains 2*N* continuous variables, *N*(*N*+1) binary variables and *N*(2N^2-4*N*+7) linear constraints. Unfortunately, as mentioned before, we failed to solve instances of 15 jobs even if we use the most performant commercial MIP codes. However, real-life instances may contain between 15 and 25 jobs to be executed over a horizon of 5 days. Thus it is obvious that we need to develop efficient heuristics to solve real-life instances.

Before presenting the heuristics developed to deal with this real-life problem, let us discuss some of the characteristics of the presented mathematical formulation.

***Proposition* 1**: A feasible solution for the model presented above always exists.

Proof: Assuming, without loss of generality, that jobs are numbered in the descending order of their width. The model is always feasible as the following solution is always feasible:

$$t_1 = max\ (a_1, a+\theta_0\ s), \tag{9}$$

$$t_j = max\ (a_j, t_{j-1}+p_{j-1});\ j = 2, ..., N, \tag{10}$$

$$r_j = max\ (0, t_j+p_j\ -d_j). \tag{11}$$

Where: θ_0 =1 if $L_1>L$ and θ_0 =0 otherwise.

This solution schedules jobs starting with the widest one down to the narrowest one. It requires one or no setup before the execution of the first job depending on whether the width of the largest job is larger than the initial width L or not. Each job is then scheduled to begin at the earliest possible time.

***Corollary* 1**: The cost of the above given solution is an upper bound on the optimal solution cost. This cost is:

$$C_u = \theta_0 S + \sum_{j=1}^{N} h_j w_j (\tau_j - a_j) + \sum_{j=1}^{N} P_j r_j \tag{12}$$

***Proposition* 2**: The CRMSP presented above is NP-hard in the strong sense.

Proof: The proof can be done by using the restriction technique (Garey & Johnson 1979, p. 63) as follows. The problem is NP-hard in the strong sense as its special case where the setup cost, set up time and inventory holding costs are nil, is NP-hard in the strong sense (Garey & Johnson, 1979, p. 237).

The next proposition shows how to determine the optimal schedule for a given sequence. This will be used within the heuristics proposed in this paper as they enumerate some possible

execution sequences and for each sequence we need to determine the corresponding optimal execution dates and costs.

***Proposition* 3**: For a given sequence, the optimal execution dates can be determined as follows. Let α be the earliest date where the mill is available and assume that jobs are numbered according to the given execution sequence σ, then the optimal start times are:

$$t_1 = max\,(a_1\,,\, a+\theta_1 s), \tag{13}$$

$$t_j = max\,(a_j\,,\, t_{j-1} + p_{j-1} + \theta_j s);\;\; j=2,...,N, \tag{14}$$

$$r_j = max\,(0,\, t_j + p_{j-1} - d_j); \qquad j=1,...,N. \tag{15}$$

Where: θ_1 =1 if $L_1>L$ and θ_1=0 otherwise

θ_j =1 if $L_j>L_{j-1}$ and θ_j=0 otherwise

The total cost corresponding to this sequence is:

$$C_\sigma = S\sum_{j=1}^{N}\theta_j + \sum_{j=1}^{N} h_j w_j (t_j - a_j) + \sum_{j=1}^{N} P_j r_j \tag{16}$$

Proof: This cost is the smallest possible cost for the given sequence as once the sequence is chosen the number of setups is fixed, the corresponding setup cost is determined and cannot be reduced. Also, the used dates are the earliest possible execution dates and as thus they minimize both the sum of the inventory holding costs and the sum of tardiness penalties.

In the following two heuristics will be developed to solve the CRMSP. The first one is a perturbation-based heuristic approach (PBHA) and the second on is a greedy randomized adaptive search procedure (GRASP).

4. A PERTURBATION-BASED HEURISTIC APPROACH TO SOLVE THE CRMSP

The proposed approach is a perturbation-based approach. Many researchers has developed perturbation heuristics and shown that this type of heuristics performs very well (see Renaud et al. 2002; Bolduc et al. 2008; and Polat et al 2015). The proposed procedure starts by applying a construction heuristic (or any other type of heuristics) to find an initial solution. It is suggested to use the new and quick heuristic, called the *Greedy multi-rule construction heuristic* (GMRCH), developed and be presented in subsection 4.1. This initial solution can be then improved by applying one or consecutively applying more than one improvement procedures. Two improvement procedures, called $I1$ and $I2$, are developed and sketched in subsection 4.2.

Afterwards, the proposed perturbation heuristic performs P iterations where in each iteration it randomly modifies (perturb) the current production sequence and then improves the perturbed sequence by consequently applying the improvement procedures $I1$ and $I2$. The resulting sequence becomes the current one. Figure 1 gives the details of this perturbation procedure.

To modify (perturb) the current sequence, the procedure repeats the following steps N times. It draws a random number from a uniform distribution between 0 and 1. If the drawn number is less than a preselected threshold ξ, it randomly chooses three jobs and circularly permutes their positions; i.e., moves the job in the first chosen position to the second position, moves the job in the second chosen position to the third one, and finally moves the job in the third position to the first one.

To apply this PBHA we have to select a value for the parameters P, the number of iterations or repetitions, and the threshold ξ. In the numerical tests reported in Section 6, ξ was set to equal 0.3 and several values of P are tested.

Use a heuristic of your choice (the GMRCH is recommended) to find an initial solution
Consecutively apply some pre-selected improvement procedures to the improve the solution

REPEAT *P* times
  REPEAT *N* times
    Draw a random number *x* from a uniform distribution between 0 and 1
    IF *x* < the chosen value of ξ THEN
      Randomly choose three different positions *p*, *q* and *r* in the current sequence
      Move the job currently in position *p* to position *q*
      Move the job currently in position *q* to position *r*
      Move the job currently in position *r* to position *p*
    END IF
  END REPEAT
  Consecutively apply the pre-selected improvement procedures to the perturbed solution
  Current sequence = resulting sequence
END REPEAT

Retain the best found solution

***Figure* 1:** The proposed perturbation-based solution procedure

4.1 *The Greedy multi-rule construction heuristic* (GMRCH)

This heuristic is a construction heuristic which constructs the production sequence in *N* steps. It starts at time $t=a$ (the earliest time the mill is available), chooses the job to execute at this time *t* and then moves forward to the time where this job is finished and chooses the next one, and so on. To select the next job to process it uses seven different conditional Greedy rules. So if the first rule is applicable, we use it to select the job to schedule. Otherwise we go to the second rule, and if the second rule is not applicable we pass to the next one, and so on.

The following notation is needed to present the heuristic rules used within the GMRCH:

k index of the job candidate to be scheduled at time t

J_t set of schedulable jobs at time t; *i.e.*, set of *j*obs j not yet scheduled and such that $a_j \leq t$

L^c width of the last processed job before time t

L^t width of the largest schedulable job at time t. Notice that L^t can be larger than, equal or smaller than L^c

m_j time slack of job j at time t; *i.e.*, $m_j=d_j-t-\theta s-p_j$ where $\theta=0$ if $L_j \leq L^c$ and $\theta=1$ if $L_j>L^c$

m_{min} minimal time slack over J_t; *i.e.*,$m_{min} = \min_{j \in J_t}(m_j)$

p_{min} minimal processing time over $\{J_t \backslash k\}$; *i.e.*,$p_{min} = \min_{j \in J_t, j \neq k}(p_j)$

J_m set of schedulable jobs j having the minimal time slack; *i.e.*, $J_m=\{j \in J_t | \ m_j=m_{min}\}$

J_{m+} the subset of J_m such that their width= min (L^c, L^t), *i.e.*, J_{m+} $\{j \in J_m | \ L_j=min(L^c, L^t)\}$

J_{tc} set of schedulable jobs at time t such that their width = L^c

J_{tt} set of schedulable jobs at time t such that their width = L^t

These job selection rules can be stated as follows. Figure 2 presents these rules in mathematical terms. For all these rules, in case of tie we choose the shortest job.

Rule 1: If there is a job k, schedulable at time t and having a time slack less than the processing times of all other jobs schedulable at t while either $L_k=L^c$ or $L^t>L^c$, choose this job k to be the one to start at t. The rationale of this rule is as follows: Not choosing this job k, implies that it will be finished after its due date. Also to make sure that this choice has a little chance to increase the total number of required setups, this rule requires that either the width of the job k be equal to L^c the largest width the mill can perform at time t (without requiring a new setup), or that L^t the largest job in the set of non scheduled jobs be larger than L^c; which implies that a future setup is needed whether we choose k or not.

Rule 2: If there is a job k, schedulable at t and having a time slack less than the processing times of all other jobs schedulable at t while both its width $L_k=L^t$ and $L^c>L^t$, choose this job k. This rule is also designed to prevent executing k after its due date. Notice that as job k has the largest width among jobs ready to be executed, we can either choose k or leave the mill idle until the arrival of a job of width = L^c. By this rule we decide not to leave the mill idle.

Rule 3: If J_{m+} (the set of jobs j schedulable at t with $m_j=m_{min}$ and $L_j=min$ $\{L^c,L^t\}$) is not empty, choose among its jobs the one which minimizes the sum of the inventory holding cost of all other jobs schedulable at t over the time needed to process the selected job. Here we choose one of the jobs having the minimal time slack. However, to reduce the chance of increasing the total number of setups, we require that the width of this job be either equal to L^t, if $L^t \leq L^c$, or equal to L^c if $L^t>L^c$. Also we require that this choice minimizes the resulting sum of holding costs of the other schedulable jobs.

Rule 4: If there is a job k among those schedulable at t and having the minimal time slack such that either its width $L_k = L^t$ or $L_k<L^c<L^t$, choose this job k. The rational of this rule is similar to the previous one.

Rule 5: If the set of jobs such that their processing time is smaller than the minimum slack and their width equals to the minimum between L^c and L^t, is not empty, choose in this set the one which minimizes the sum of the inventory holding cost of all other schedulable jobs over the time needed to process the selected job. The rational here is similar to that of rule 3.

Rule 6: If $L^t \geq L^c$ and J_{tc} (the set of schedulable jobs having a width = L^c), is not empty, then choose the job $k \in J_{tc}$ having the smallest slack.

Rule 7: Among the jobs having a width = L^t, choose the job having the smallest time slack.

1- If there is a job $k \in J_t$ such that $m_k<p_j \forall j \in J_t$ and $j \neq k$ and ($L_k=L^c$ or $L^t>L^c$) then schedule this job k. If there is more than one such job choose the shortest job.
2- If there is a job $k \in J_t$ such that $m_k<p_j \forall\ j \in J_t$ and $j \neq k$ and $L^c > L^t=L_k$ then schedule this job k. If there is more than one such job choose the shortest job.
3- If there are some jobs $k \in J_m$ such that $L_k= min\ (L^c, L^t)$ then select among them the one that minimizes: $p_k \sum_{j \in J_t, j \neq k} h_j w_j$. In case of tie chose the shortest job.
4- If there are some jobs $k \in J_m$ and ($L_k=L^t$ or $L_k<L^c<L^t$) then schedule the one that minimizes: $p_k \sum_{j \in J_t, j \neq k} h_j w_j$. In case of tie chose the shortest job.
5- If there are some jobs $k \in J_t$ such that $p_k<m_{min}$ and $L_k= min\ (L^c, L^t)$ then schedule the one that minimizes: $p_k \sum_{j \in J_t, j \neq k} h_j w_j$. In case of tie chose the shortest one.
6- If $L^t \geq L^c$ and J_{tc} is not empty then choose the job $k \in J_{tc}$ having the smallest slack.
7- If J_{tc} is empty or $L^t < L^c$ then choose the job $k \in J_{tt}$ having the smallest slack.

***Figure* 2:** The seven decision rules of the Greedy multi-rule construction heuristic

4.2 *Sequence improvement procedures I*1 *and I*2

These sequence improvement procedures are simple hill climbing procedures. The difference between these two procedures is in their searched neighbourhood. The neighbourhood of *I*1 is all the neighbour sequences where only one job is moved to a different position. The neighbourhood of *I*2 is all the neighbour sequences where only one pair of jobs exchanges their position in the current sequence. A pseudo code of these two sequence improvement procedures is depicted in Figure 3. In this Figure, *p*(*j*) stands for the position of job *j* in the sequence.

5. *A Greedy Randomized Adaptive Search Procedure* (GRASP)

The second heuristic to be tested is a greedy randomized adaptive search procedure (GRASP). Many researchers developed and tested this heuristic and indicate that it produces very good solutions (see for example Resende et al. 2014). This heuristic generates a number of production sequences, improves each sequence and retains the best one. Figure 4 gives the main procedure and the sub-procedures of the proposed GRASP.

Each sequence is obtained by a construction heuristic which starts at time $t=a$ (the earliest time where the rolling mill is available), calculate for each job i schedulable at time t, the incremental cost C_i of scheduling it at t, build a restricted candidate list (RCL) including all jobs such that $C_i \leq C_{min}+\alpha(C_{max}-C_{min})$, and randomly choose among the these jobs the one to schedule. Then we move to the time moment where this job should finish and choose the next one, and so on.

```
PROCEDURE I1
    WHILE improvement=TRUE
        improvement=FALSE
        FOR j=1 to N
            FOR all positions p except p(j)
                Calculate D_jp, the cost change if j is moved to position p
            NEXT p
            IF D_jq =min_p {D_jp}<0 THEN
                Improvement=TRUE: Move job j to the corresponding position q
            END IF
        NEXT j
    WHILE END

PROCEDURE I2
    WHILE improvement=TRUE
        improvement=FALSE
        FOR j=1 to N
            FOR l=1 to N and l≠j
                Calculate D_jl, the cost permuting the positions of jobs j and l
            NEXT l
        NEXT j
        IF D_ik =min_{j,l} {D_jl}<0 THEN
            Improvement=TRUE: Permute the positions of jobs i and k
        END IF
    WHILE END
```

***Figure* 3:** Pseudo code of the improvement procedures *I*1 and *I*2

As indicated by Resende and Ribeiro (2008), the way to choose the value to attribute to α has an effect on the quality of the produced solution. They suggest that chosing α at random from a uniform probability distribution leads to good results. This method is used in this GRASP implementation. Another important parameter is *G*, the number of iterations to use. In section 6, several values of *G* are tested.

GRASP main procedure
  REPEAT *G* times
    Build a production sequence σ (PROCEDURE SEQUENCE BUILDING)
    Improve the built sequence until getting a local optimum σ_{Local} (PROCEDURE SEQUENCE IMPROVEMENT)
    IF the cost of σ_{Local} is smaller than the cost of the best found sequence σ_{best} Then $\sigma_{best}=\sigma_{Local}$
  END REPEAT
  RETURN σ_{best}

PROCEDURE SEQUENCE BUILDING
  REPEAT *N* times
    Establish the list of jobs that can be processed next
    Evaluate the incremental cost C_i of each of these jobs
    Determine C_{min} and C_{max}, the minimum and maximum of the calculated incremental costs
    Randomly choose a value for α from a uniform distribution between 0 and 1
    Put in the restricted candidate list (*RCL*) all jobs *i* such that $C_i \leq C_{min}+\alpha(C_{max}-C_{min})$
    Randomly choose and schedule a job from the *RCL*
  END REPEAT

PROCEDUR SEQUENCE IMPROVEMENT
  Consecutively apply the improvement procedures *I*1 and *I*2

***Figure* 4:** the developed GRASP

6. HEURISTICS PERFORMANCE

6.1 *Test instances*

Three sets of one hundred instances each were randomly generated and solved to assess the performance of the proposed heuristics. The instances of these three sets have respectively 10, 15 and 25 jobs. These instances have the same characteristics of the real-life problem observed in the studied Aluminium rolling facility. Table 1 presents parameters values used to generate the test instances. For parameters having several possible values, the probability of choosing one of these values is the same for all of them.

All test instances were solved using the commercial MIP code GORUBI 6.5.1 with a run time limit of 3,600 seconds as well as by the perturbation-based heuristic approach presented in section 4 with several values of *P*, and by the Greedy randomized adaptive sampling procedure presented in section 5 with several values of *G*.

***Table* 1:** Parameters values for the randomly generated test-instance

Parameter	Values
Set up cost S	300
Setup time s	1.5
Initial width L	57
Tardiness penalty P_j	30
Inventory holding cost h_j	0.025
Job width L_j	41,45,49,53,57,61
Job weight w_j	20,25,30,..,225
Job arrival date a_j	0,6,12,18
Job due date d_j	6,12,18,24
Processing time p_j	1,2,3,4,5

6.2 *Results obtained by solving the mathematical model*

We tried to solve the test problems to optimality using the MIP code GUROBI 6.5.1 and a computer equipped with a 3.50 GHz, 8 cores Intel Xeon processor. All the 10-job instances were solved to optimality within an average of 61.77 second of computation time.

However, it was not possible to solve the 15-job instances to optimality. Actually, after 10 days of computation the first problem was not solved. Thus it was decided to put a limit of 3,600 seconds (one hour) on the run time. No 15-job instance was solved to optimality within this time limit and none of the solutions obtained after one hour was better than the best solutions obtained by the tested heuristics. For these 15-job instances, the average deviation of the solution obtained by GUROBI after one hour from the best heuristic solution was 1.07%. The minimum deviation was 0 % and the largest deviation was 10.89% with a standard deviation of 1.91%.

Consequently, for the 15-job and 25-job instances, it was decided to assess the performance of the tested heuristics by comparing their solutions to the best solution among all the obtained heuristic solutions. The tested heuristics were executed on a micro computer equipped with a 2.00 GHz Intel core 2 Duo processor.

6.3 *Heuristic solutions for the 10-job instances*

As mentioned before all the one hundred instances with 10 jobs were solved to optimality within an average computation of 61.77 seconds by GUROBI 6.5.1 and a computer equipped with a 3.50 GHz, 8 cores Intel Xeon processor. So we can use these optimal solutions to evaluate the performance of the proposed heuristics.

Table 2 gives the results obtained by the perturbation approach for several values of *P*, the number of perturbation iterations. The table shows that the optimal solution of all the 10-job instances were obtained with *P*= 20, 50 and 100.

***Table* 2:** Results for the 10-job instances as obtained by the perturbation approach

P=	1	10	20	50	100
Average percentage deviation from the optimal solution	2.43%	0.07%	0.00%	0.00%	0.00%
Minimal percentage deviation from the optimal solution	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the optimal solution	51.89%	5.53%	0.00%	0.00%	0.00%
Standard deviation	6.56%	0.56%	0.00%	0.00%	0.00%
Number of times the optimum was found	69	98	100	100	100
Average computation time (seconds)	0.02	0.16	0.34	0.84	1.70

Table 3 gives the results obtained by the greedy randomized adaptive search procedure (GRASP) presented in section 5 for different values of *G* (the number of sampled solutions).

From these tables we can see that the GRASP with G =100 was able to produce the optimal solution for all and every 10-job test-instance. For the other tested values of G, there were always some instances where the GRASP failed to find the optimal solution.

***Table* 3:** Results for the 10-job instances as obtained by the GRASP

G=	1	10	20	50	100
Average percentage deviation from the optimal solution	3.067%	0.002%	0.007%	0.012%	0.00%
Minimal percentage deviation from the Optimal solution	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the optimal solution	34.23%	0.23%	0.49%	1.25%	0.00%
Standard deviation	6.79%	0.02%	0.05%	0.12%	0.00%
Number of times the optimum was found	67	98	99	99	100
Average computation time (seconds)	0.02	0.16	0.34	1.03	1.67

6.4 *Heuristic solutions for the 15-job instances*

As mentioned above, it was not possible to obtain the optimal solutions for the 15-job test-instances. So it was decided to compare the solutions obtained by the different versions of the two proposed heuristics to the best solution among all the obtained heuristic solutions.

Table 4 summarizes the results obtained by the perturbation approach with different values of *P*, the number of perturbation iterations. From this Table, we can see that increasing the number of iterations *P* decreases the percentage deviation from the best heuristic solution. However, the computation time also significantly increases with *P*. Interestingly, we found that with *P*=100, we obtain the best solution for all and every instance. We also notice that increasing *P* from 1 to 10 improves the solution of 51 of the 100 instances while increasing *P* from 10 to 20 improves or further improves the solution of only 9 instances. Also, going from *P*=20 to *P*=50 improves the solution of 4 other instances and from *P*=50 to *P*=100 improves the solution of another 4 instance. For 42 of the 100 test-instances, the solution obtained with *P*=1 was never improved.

In order to see if we can further improve the results if we increase the value of *P*, the perturbation heuristic was applied with *P*=1000. This produced exactly the same solutions as with *P*=100. Obviously, the computation time increased proportionally to an average of 69.59 seconds per instance.

***Table* 4:** Results obtained by the perturbation approach for the 15-job instances

P=	1	10	20	50	100
Average percentage deviation from the best found solution	2.77%	0.19%	0.12%	0.03%	0.00%
Minimal percentage deviation from the best found solution	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the best found solution	18.55%	4.64%	5.22%	1.43%	0.00%
Standard deviation	4.11%	0.70%	0.59%	0.16%	0.00%
Number of instances for which the best solution was found	42	87	92	96	100
Average computation time (seconds)	0.07	0.72	1.42	2.41	4.97

Table 5 summarizes the results obtained by applying the GRASP with different values of *G* the number of sampled solutions. These results show that with *G* = 100 the GRASP was able to find the best solution for all and every 15-job instance. Again, to see if increasing the number of sampled solutions G can lead to better results, the GRASP was applied with *G*=1000. This produced exactly the same solutions as with *G*=100 but the computation time jumped to an average of 72.59 seconds per instance.

***Table* 5:** Results for the 15-job instances as obtained by the GRASP

G=	1	10	20	50	100
Average percentage deviation from the best found solution	3.16%	0.17%	0.09%	0.02%	0.00%
Minimal percentage deviation from the best found solution	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the best found solution	18.62%	3.53%	2.63%	1.06%	0.00%
Standard deviation	4.48%	0.60%	0.36%	0.12%	0.00%
Number of instances for which the best solution is found	39	86	91	97	100
Average computation time (seconds)	0.07	0.72	1.43	3.54	7.13

6.5 Heuristic solutions for the 25-job instances

Hereafter, the solutions obtained by the different versions of the two proposed heuristics are compared to the best solution among all the obtained heuristic solutions. Table 6 summarizes the results obtained by the perturbation approach with different values of *P*, the number of perturbation iterations. From this Table we can see that increasing *P* decreases the percentage deviation from the best heuristic solution but the computation time also increases. The Table shows that the perturbation approach with *P* = 200 was able to find the best solution for 98 of the one hundred 25-job instances. Again, in order to see if we can improve the results if we further increase the value of *P*, the perturbation heuristic was applied with *P*=1000. This produced the

same solution as with P=200 for every instance. Obviously, the computation time increased proportionally and reaches an average of 428.84 seconds per instance.

***Table* 6:** Results obtained by the perturbation approach for the 25-job instances

P=	1	10	20	50	100	150	200
Average percentage deviation from the best found solution	1.83%	0.30%	0.23%	0.07%	0.05%	0.03%	0.00%
Minimal percentage deviation from the best found solution	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the best found solution	7.46%	5.31%	4.76%	1.23%	0.87%	0.87%	0.00%
Standard deviation	1.97%	0.79%	0.59%	0.22%	0.17%	0.13%	0.00%
Number of instances for which the best solution was found	24	70	70	88	95	98	100
Average computation time (seconds)	0.39	4.32	8.24	21.06	41.92	61.35	83.68

Table 7 summarizes the results obtained by applying the GRASP with different values of G. These results show that with G = 200 the GRASP was able to find the best solution for 98 of the one hundred 25-job instances. Again, to see if increasing the number of sampled solutions G can lead to better results, the GRASP was applied with G=1000. No better solutions were obtained. Obviously, the computation time also increases and reaches an average of 431.15 seconds per instance. Comparing the results given in Tables 6 and 7 indicates that also for the 25-job instances the two developed heuristics produced similar results in similar computation times.

***Table* 7:** Results for the 25-job instances as obtained by the GRASP

G=	1	10	20	50	100	150	200
Average percentage deviation from the best found solution	2.39%	0.23%	0.16%	0.10%	0.04%	0.02%	0.02%
Minimal percentage deviation from the best found solution	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Maximal percentage deviation from the best found solution	9.34%	3.77%	2.77%	1.51%	1.38%	0.87%	0.87%
Standard deviation	2.38%	0.58%	0.41%	0.27%	0.18%	0.12%	0.12%
Number of instances for which the best solution is found	14	68	75	84	94	97	98
Average computation time (seconds)	0.46	4.70	8.57	21.46	42.97	62.78	84.23

7. CONCLUSIONS

This paper presents a new formulation of the *cold rolling mill scheduling problem* (CRMSP). This formulation allows solving instances with 10 jobs but failed to solve instances of the problem with 15 jobs. Thus the paper proposes *perturbation-based heuristic solution approaches* and a *greedy randomized adaptive search procedure* (GRASP) to solve real-life instances. The perturbation-based heuristic is an iterative approach that uses a new construction heuristic, called the *Greedy multi-rule construction heuristic* (GMRCH), to obtain an initial solution and then uses two local search improvement heuristics to improve the obtained solution. Afterwards, it perform a number of iterations where each one consists of perturbing (modifying) the current solution and improving the perturbed solution by applying the same local search improvement heuristics. The proposed GRASP uses the same local search improvement heuristics to improve the randomly sampled solutions.

Three sets of one hundred test-instances each were used to evaluate the performance of the proposed solution methods. The number of jobs is 10 for the first instance-set, 15 for the second, and 25 for the third set. It is shown that the proposed perturbation approach using the GMRCH as well as the GRASP was able to obtain the optimal solution for all the 10-job instances within less than 2 seconds of computation time. For the 15-job instances, where it was not possible to obtain the optimal solutions, the perturbation approach with one hundred iterations (P=100) produced the best results for all instances within less than 5 seconds of computation time. The proposed GRASP with one hundred sampled solutions (G=100) was also able to find the best solutions for all instances within an average computation time 7.13 seconds. For the 25-job instances the perturbation heuristic with P=200 produced the best solution for all instances while the GRASP with G=200 produced the best solutions for 98 of the 100 instances. The average computation time in both case was about 84 seconds.

However, although the proposed solution approach is able to produce quite good solutions, it is still necessary to try to develop more efficient heuristic methods. It is also necessary to develop ways to solve larger instances to optimality.

Acknowledgement

This research work was partially supported by Grant OPG0036509 from the National Science and Engineering Research Council (NSERC). This support is gratefully acknowledged.

REFERENCES

As'ad, R. & Demirli, K. (2010). Production scheduling in steel rolling mills with demand substitution: Rolling horizon implementation and approximations. *International journal of Production Economics*, 126, 361-369.

Bolduc, M.-C., Renaud, J., Boctor F. F. & Laporte, G. (2008). A perturbation metaheuristic for the vehicle routing problem with private fleet and common carriers, *Journal of the Operational Research society***,** 59, 776-787.

Chen, A. L., Yang, G.K. &. Wu, Z.M. (2008). Production scheduling optimization algorithm for the hot rolling process. *International Journal of Production Research*, 46, 1955-1973.

Cowling, P. (2003). A Flexible decision support system for steel hot rolling mill scheduling. *Computers and Industrial Engineering*, 45, 307-321.

De Ladurantaye, D., Gendreau, M. & Potvin, J-Y. (2007). Scheduling a hot rolling mill. *Journal of the Operational Research Society*, 58, 288-300.

Dixit, U.S. & Dixit, P.M. (2000). Application of fuzzy set theory in scheduling of a tandem Cold-rolling mill. *Journal of Manufacturing Science and Engineering*, 122, 494-500.

Garey, M.R., & Johnson, D.S. (1979). *Computers and Tractability*. New York; W.H. Freeman and Company.

Lopez, L., Carter, M.W., & Gendreau M. (1998). The Hot strip mill production scheduling problem: A Tabu Search Approach. *European Journal of Operational Research,* 106, 317-335.

Mayrand, E., Lefrançois, P., Kettani O. & Jobin, M.H. (1995). A Genetic search algorithm to optimize job sequencing under technological constraint in a rolling-mill facility. *Operations Research Spektrum*, 17, 183-191.

Nicholls, M.G. (1994). Optimizing the operations of an ingot mill in an aluminium smelter. *Journal of the Operational Research Society*, 45, 987-998.

Pinedo, M.L. (2012). Scheduling: theory, algorithms and systems. Fourth edition, Springer.

Polat, O., Kalayci, C.B., Kulak, O. & Günther H.O. (2015). A perturbation based variable neighborhood search heuristic for solving the Vehicle Routing Problem with Simultaneous Pickup and Delivery with Time Limit, *European Journal of Operational Research*, 242, 369-382.

Renaud, J., Boctor, F. F. & Laporte, G. (2002). Perturbation heuristics for the pickup and delivery traveling salesman problem, *Computers and Operations Research*, 29, 9, 1129-1141.

Resende, M.G.C. & Ribeiro, C.C. (2008). Greedy Randomized adaptive search procedures: advances and applications, in Handbook of metaheuristics, Gendreau M. and Potvin J.Y. (eds.) Springer.

Resende, M.G.C. & Ribeiro, C.C. (2014). Greedy Randomized adaptive search procedures, in search methodologies, Burke E. K. & Kendall G. (eds.) Springer.

Sasidhar, B. & Achary K.K. (1991). A Multiple arc network model of production planning in a steel mill. *International Journal of Production Economics*, 22, 195-202.

Stauffer, L., & Liebling T.M. (1997). Rolling horizon scheduling in a rolling-mill. *Annals of Operations Research*, 69, 1, 323-349.

Tang, L., Liu, J. Rong, A. & Yang Z. (2000). A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron and Steel Complex. *European Journal of Operational Research*, 124, 267-282.

Wang, D.D., Tieu, A.K., De Boer, F.G., Ma, B. & Yuen, W.Y.D. (2000). Towards a heuristic optimum design of rolling schedules for tandem cold rolling mills. *Engineering Applications of Artificial Intelligence*, 13, 397-406.

Yang, J.M., Che, H.J., Dou, F.P. & Zhou, T. (2008). Genetic algorithm-based optimization used in rolling schedule. *International Journal of Iron and Steel Research*, 15, 18-22.

Zhao, J., Liu, Q. L., & Wang, W. (2008). Models and algorithms of production scheduling in tandem cold rolling. Acta Automatica Sinica, 34(5), 565-573.