March 22, 2007

RT0718

Computer Science 9 pages

Research Report

A heuristic method for scheduling problems with position constraints

Takayuki Yoshizumi and Toshiyuki Hama

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A heuristic method for scheduling problems with position constraints

Abstract

Local search is widely used for optimization problems such as scheduling problems to improve the quality of solutions. In some cases, however, neighborhood operations at an early stage prevent a solution from being improved in a later stage. In this report, we present a heuristic method that executes neighborhood operations considering the later stages by using a function that estimates the objective value at the end of the search. Using this method, we can avoid neighborhood operations that make the objective value a lot better at that point, but which have adverse effects on the later stages. As a result, we can get better schedules. Experimental results show that our method makes better schedules for about half of the instances in a real-world scheduling problem.

1. Introduction

For hot strip mill scheduling in the steel industry, it is difficult to automatically generate schedules with algorithms because there are many constraints due to limitations of the facilities for satisfying the quality requirements for the products. Recently we proposed a method that adopts a coarse-to-fine approach to make schedules, which can be used in hot strip mills [1]. This method first generates an initial solution by integer programming and other means, and then improves the initial solution by local search. The problem has position constraints, which limit the positions of some items. For example, certain items must be located between the 30-th and 60-th positions from the beginning of a schedule. In some cases, these position constraints make the performance of local search worse, because the neighborhood operations in an early stage sometimes prevent a solution from being improved in a later stage.

We present a heuristic method that executes neighborhood operations considering the later stages by using a function that estimates the objective value at the end of the search. First, the tightness of a schedule is quantified by mapping the state of the schedule into a graph. Then we estimate the number of items to be inserted into the schedule by using a recurrence formula based on the graph. Using this method, we can avoid neighborhood operations that make the objective value much better at that point, but which have adverse effects in the later stages.

2. Hot strip mill scheduling problem

The hot strip mill (HSM) scheduling problem [1] is a problem to form the largest possible milling sequences of thick steel plates, which are called slabs, considering various kinds of constraints. There are position constraints limiting the positions of some slabs. Example conditions are shown below.

- Those slabs that meet Condition A must be located no later than the 60-th from the beginning.

- Those slabs that meet Condition B must be located between the 30-th and 60-th positions.
- Those slabs that meet Condition C must be located after the 60-th from the beginning.

The objective value of a schedule is calculated based on the attributes and similarities of adjacent pairs of slabs in the schedule, as well as on the length of the schedule (the number of slabs in the schedule).

We can generate a schedule for HSM with the method that we proposed [1]. This method first generates an initial solution by integer programming and other techniques, and then improves the initial solution by local search. The length of an initial solution is relatively rather short compared to the final schedule, because an initial solution focuses on those slabs that can be key slabs for entire schedules. Therefore, it is necessary for the local search to insert as many slabs as possible from the slab pool into an initial schedule.

The basic neighborhood operation of the local search for the HSM problem can be defined as an insertion of a slab from the slab pool into any position of a schedule. There are many positions into which a slab can be inserted and the objective value after insertion varies depending on where it is inserted. Figure 1 includes a flowchart of the local search for the HSM problem.

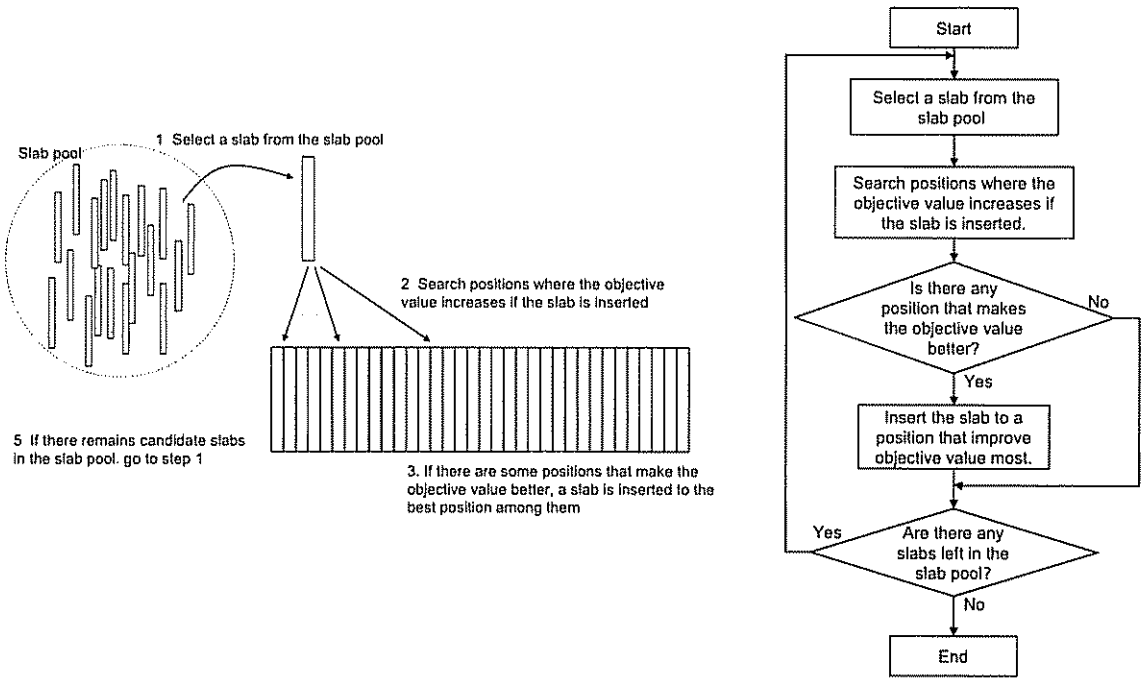

Figure 1: The algorithm of the local search

In most cases, this local search can make the length of the schedule much longer. In some cases, however, this local search does not work well due to position constraints. As shown at Figure 2, a

position constraint limits the number of slabs that can be inserted between the beginning and the last slab of the constraint while still meeting the conditions of the position constraint. (If an algorithm for generating initial solutions ensures that an initial solution will always satisfy positions constraints, then we do not have to consider the lower bounds of position constraints in this local search. We discuss only the upper bounds of position constraints from this point. We will later discuss the cases when the initial solutions violate the lower bounds.)

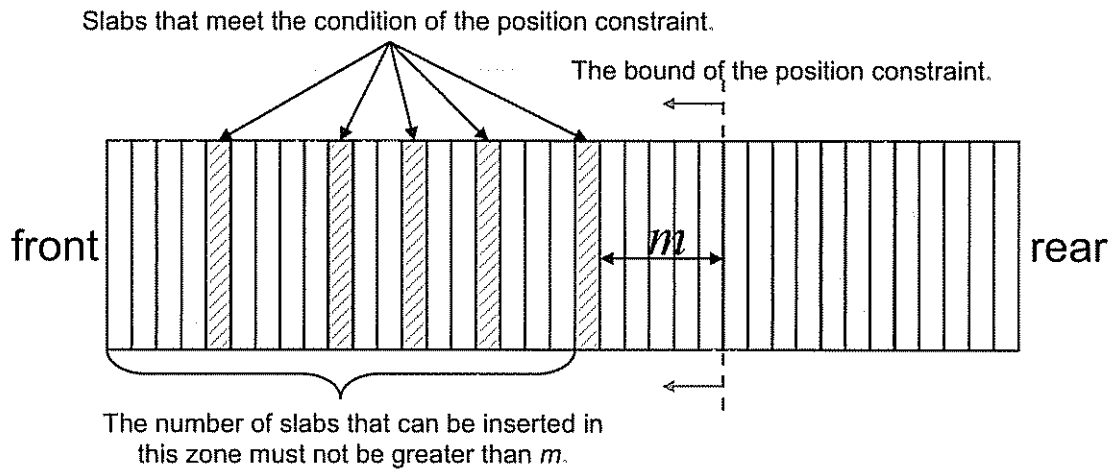

Figure 2: Effect of a position constraint

In general, a schedule is tighter when the range that is imposed by a position constraint is wider and the number of slabs that can be inserted is smaller. The tightness, which is the range imposed by a position constraint and the number of slabs to be inserted, may be changed by a neighborhood operation, which is to say by the insertion of a slab. Figure 3 shows an example of the insertion of an unconstrained slab and Figure 4 shows that of a constrained slab. These two examples show the differences of the tightness of schedules between inserting before/after the last constrained slab. The position to which a slab is inserted has a great impact on the tightness of a schedule, especially when inserted slab is a constrained slab.

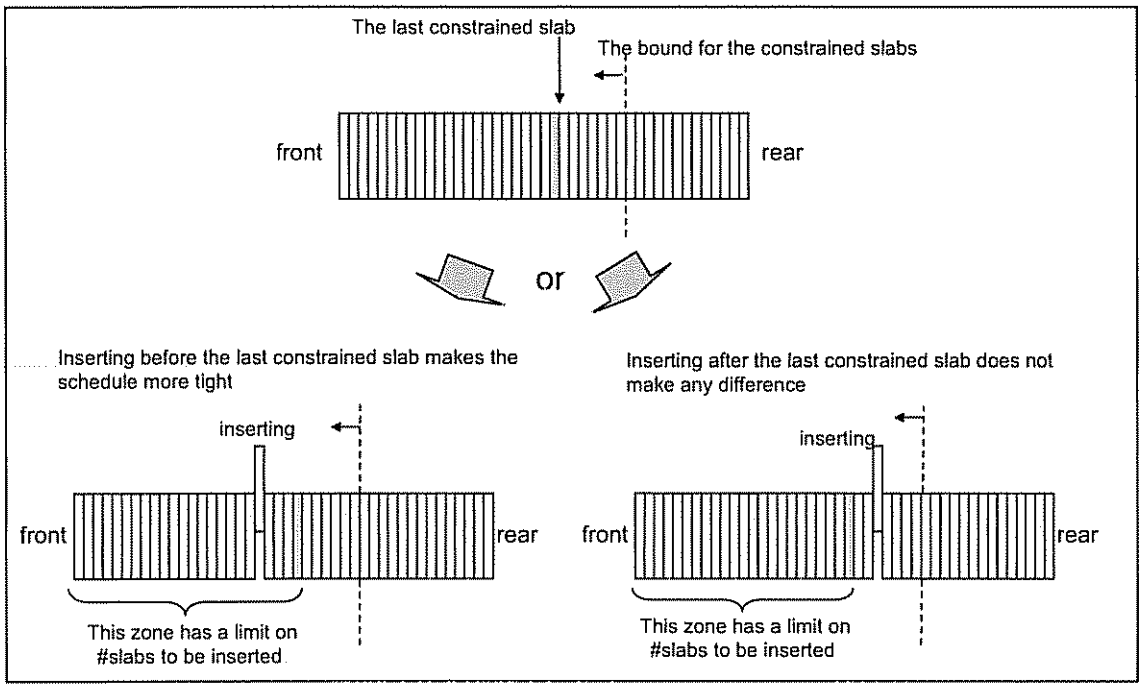

Figure 3: Effect of the insertion of an unconstrained slab.

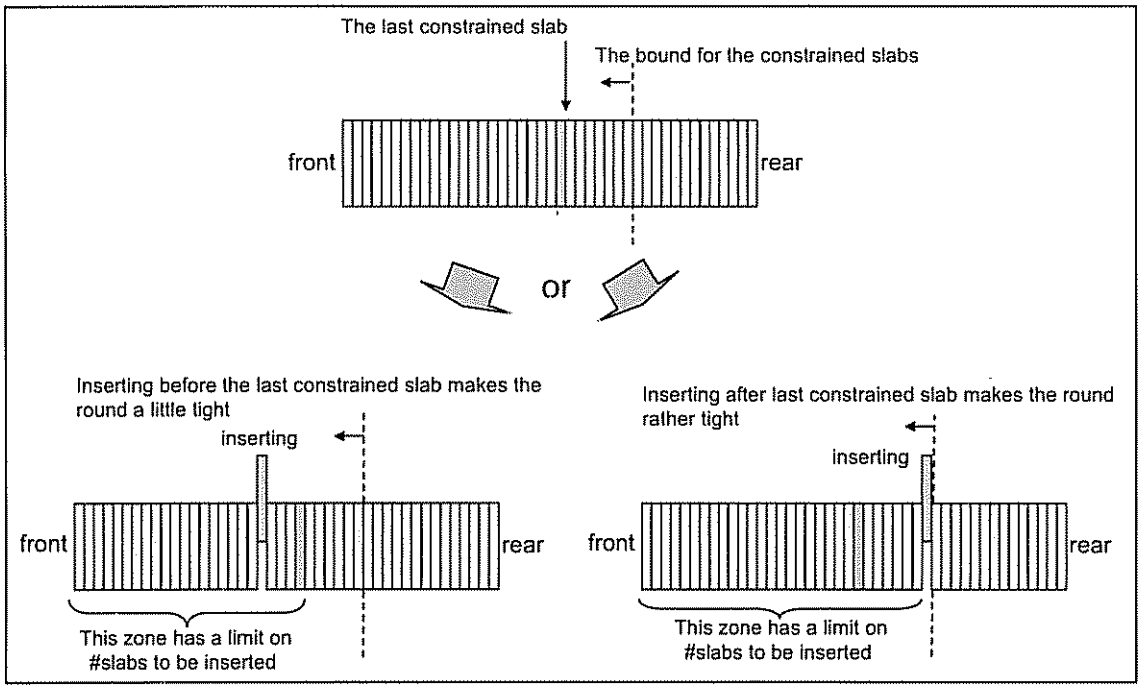

Figure 4: Effect of the insertion of a constrained slab.

The naïve local search inserts a slab at the position that improves the objective value most, even if

that insertion makes the schedule too tight. Sometimes a schedule becomes too tight in the early stages of local search. In such cases, the final schedule will not be good because only a few insertions can be done in the later stages of the local search.

A neighborhood operation that does not improve the objective value as much but which does not make the schedule too tight may lead to a better final schedule. In general, however, local searches cannot do this.

3. Proposed method

In this chapter, we explain how to quantify the tightness of a schedule and use this notion of tightness to improve the effectiveness of local searches.

3.1. Quantifying tightness of a schedule

Methods that quantify the tightness of a schedule and use this metric for scheduling algorithms are called Constraint-Directed Search [2]. Several methods for job-shop scheduling problem are known [3]. Since the method of quantifying a schedule depends on the structure of each problem, the methods for job-shop scheduling cannot be applied directly to the HSM scheduling problem.

For the HSM problem, as shown in Figure 2, the tightness of a schedule can be represented as a zone that has a restriction on the number of slabs that can be inserted and that number. In general, however, there are multiple position constraints. Therefore we should appropriately quantify multiple position constraints at the same time.

Suppose that the tightness of a schedule due to its position constraints is mapped into a graph such as shown in Figure 5. This example shows a schedule that has two position constraints mapped into a graph. The horizontal axis represents the position from the beginning, and the vertical axis represents the number of slabs that can be inserted. Since all of the position constraints must be met, the minimum value has to be met where there are two or more constraints.

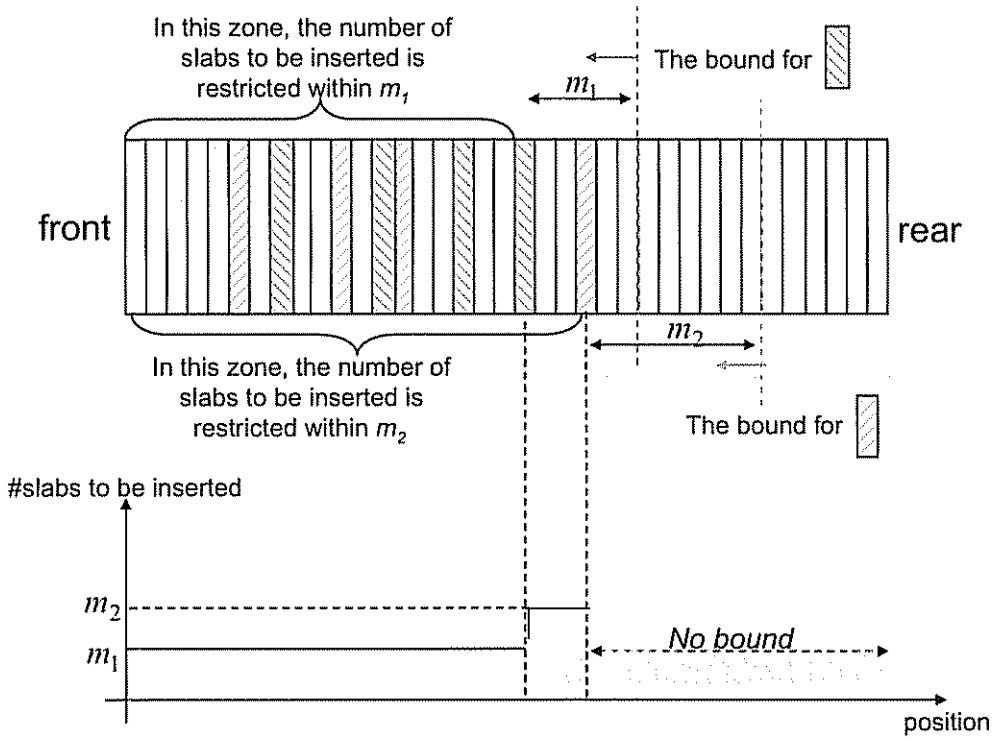

Figure 5: A mapping from a schedule to a graph

This mapping method enables us to quantify even those schedules that have multiple position constraints as a graph. In general, we will obtain a stair-like graph from a schedule as shown at Figure 6.

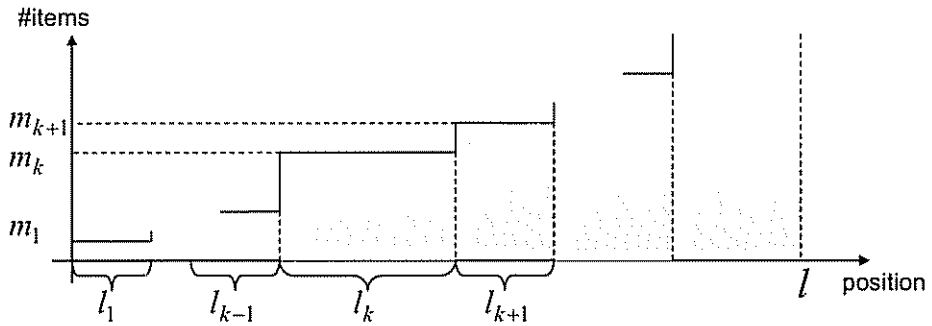

Figure 6: A representation of position constraints as a graph

3.2. Calculating the expected value using a recurrence formula

We want to derive a reasonable index that the local search can use from such a stair-like graph. The local search can be performed while looking ahead to see if the expected value of the number of

slabs that can be inserted is available in addition to the objective value. Using the example shown in Figure 6, we present a method that estimates the number of slabs to be inserted from the graph that represents the tightness of each schedule.

For simplicity, any slab in the slab pool can be inserted into a position of a schedule with a constant probability, regardless of the attributes of the slab and the position. Let p be the probability. (For an HSM schedule, the probability that a slab randomly selected from the slab pool can actually be inserted into a position in the schedule is typically less than 1 percent. So we can assume that $p << 1$.)

In the local search, the probability that a slab is inserted into the zone l_k is $\left(1-(1-p)^l\right)\frac{l_k}{l} \approx pl_k$, where l represents the length of a schedule, which is the number of slabs in the schedule. (The approximation of $(1-p)^l \approx 1-pl$ can be used as long as $p << 1$.)

Letting the length of l_k after trying the nth slab in the slab pool be l_k^n, the following recurrence equation holds:

$$l_k^{n+1} = l_k^n + pl_k^n = (1+p)l_k^n$$

Therefore the general term becomes:

$$l_k^n = l_k(1+p)^n$$

After trying all of the remaining slabs in the slab pool, the expected value of the number of slabs to be inserted is

$$l_k^N - l_k = l_k(1+p)^N - l_k \approx Npl_k,$$

where N is the number of remaining slabs.

Considering m_k, which is the limitation on the number of slabs that can be inserted due to the position constraints on the zone of l_k, and those slabs that will be inserted in earlier positions, the actual expected value of the number of slabs to be inserted into the zone of l_k becomes:

$$s_k = \min\left(m_k - \sum_{l=1}^{k-1} s_l, Npl_k\right)$$

The value of s_k can be obtained by calculating from $k=1$ one by one. Thus the expected value of the number of slabs to be inserted in the complete schedule becomes $s = \sum_k s_k$.

Using this value s, the local search can perform a search while also considering future states. As a result, the performance of the local search can be improved.

3.3. Handling lower bounds of position constraints

Taking anticipated slab insertions for the local search into consideration, the initial solutions do not

always have to meet the lower bounds of position constraints. As shown at Figure 7, a schedule that does not meet the lower bound of a position constraint can be acceptable as an initial solution. By allowing violations on lower bounds, the final schedules may be better.

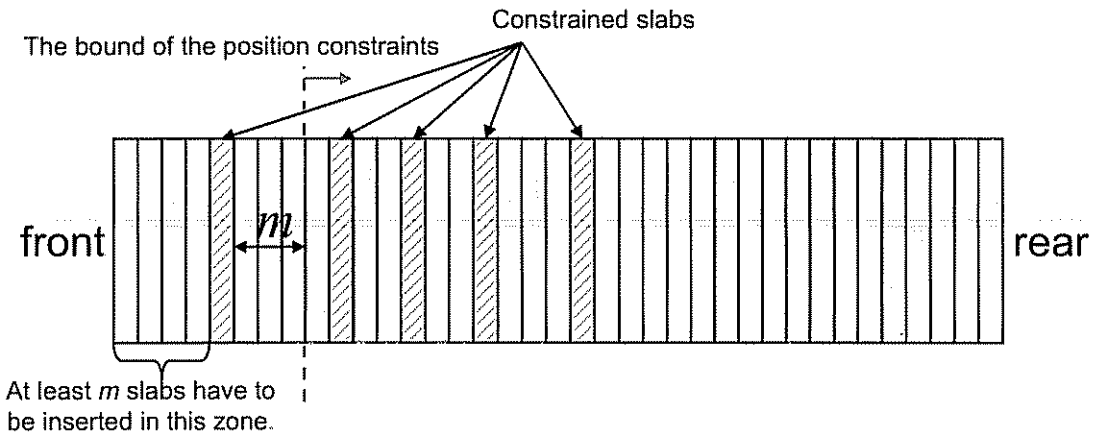

Figure 7: An example of the lower bound of a position constraint

For a schedule that has constraint violations, some penalty value can be imposed on its objective value. A neighborhood operation that eliminates or reduces violations improves the objective value a great deal. Therefore violations on lower bounds are naturally eliminated in the early stages of the local search.

Those neighborhood operations that eliminate or reduce violations in the early stages of the local search can make the final solution quality worse. However, we calculate the expected value and can exploit it as for the upper bounds already described. In the example shown in Figure 7, the expected value of the number of slabs to be inserted into the zone l can be calculated as Npl using the same method as for the upper bound case. When $Npl \geq m$ holds, the violations of the lower bounds are naturally eliminated by the end of the local search process without adding any penalty values to the objective value. Therefor it is possible to avoid inappropriate neighborhood operations in the early stages of the local search by not adding the penalty values to the objective value.

4. Experimental results

We conducted experiments on about 100 instances of the HSM scheduling problem. For 50% of the instances, we obtained better schedules using the proposed method. For 40% of the instances, there were no differences, and in the remaining 10% of the instances, the schedules become a little worse.

In general, heuristic methods have no bounds for solution qualities. We can even intentionally create pathological instances such that the heuristic methods will not work well. Considering this characteristic of heuristic methods, these experimental results show the high effectiveness of the proposed method.

5. Conclusions

In this report, we present a method that maps the tightness of a schedule to a graph and estimates the number of slabs to be inserted from the graph using a recurrence formula. This method allows us to execute a local search while considering the future states.

The experimental results show that the proposed method can improve the performance of the local search. We can apply this particular method only to the HSM scheduling problem. However, we are now trying to apply this method to more general problems such as the job-shop scheduling problem.

[1] http://www.research.ibm.com/trl/projects/optimization/hsm.htm

[2] Beck, J.C., and Fox, M.S., "A Generic Framework for Constraint-Directed Search and Scheduling", AI Magazine, Vol. 19, No. 4, Winter 1998, pp. 101-130.

[3] Sadeh, N., "Look-ahead Techniques for Micro-opportunistic Job-Shop Scheduling", Ph.D. thesis, CMU-CS-91-102, Carnegie-Mellon University.