IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 20, 2021, accepted January 30, 2021, date of publication February 10, 2021, date of current version April 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3058328

Very Large-Scale Neighborhood Search for Steel Hot Rolling Scheduling Problem With Slab Stack Shuffling Considerations

YARONG SHI AND SHIXIN LIU, (Member, IEEE)
The State Key Laboratory of Synthetical Automation for Process Industries, College of Information Science and Engineering, Northeastern University, Shenyang 110819, China

Corresponding author: Shixin Liu (sxliu@mail.neu.edu.cn)

This work was supported in part by the National Key R&D Program of China under Grant 2017YFB0304201.

ABSTRACT We study a steel hot rolling scheduling (HRS) problem considering slab stack shuffling (SSS), which is a kind of key issues in the steel strip production. The HRS problem is to select suitable slabs for a predefined sequence of hot rolling slots, from their respective candidate slab sets so that the number of shuffling operations of slabs is minimized. Different from the most researches, we consider a situation where there are intersections among candidate slab sets and shuffled slabs will not return original stacks. Basing on a special index method of slabs, we present an integer linear programming (ILP) to compute approximation solutions of the problem. In order to improve the solutions obtained by the ILP model, we propose a very large-scale neighborhood (VLSN) search algorithm. In the VLSN search process, the solutions of HRS problems are improved by an iteration procedure that partial solutions are interactively destroyed and repaired. In each iteration, partitioned hot rolling slots correspond to a sub-problem. In the repair operations, we propose an efficient branch-and-bound algorithm for solving sub-problem. The computational results on a number of different scale simulated instances show that the VLSN search algorithm is efficient for solving this kind of HRS problems.

INDEX TERMS Branch-and-bound algorithm, local search, steel hot rolling schedule, slab stack shuffling, very large-scale neighborhood search.

I. INTRODUCTION

The steel hot rolling scheduling (HRS) is one of key tasks in operation management of steel production processes, which has an important impact on product quality and production efficiency of hot rolling processes. The HRS problem is a complex combinatorial optimization problem. It rises from an industrial problem where many practical constraints must be considered and thus hard to solve. Therefore, it has been studied extensively.

In early studies, the HRS problem was mainly converted into a kind of traveling salesman problems (TSP) or vehicle routing problems (VRP) to solve. The travel cost between nodes in the TSP/VRP corresponds to the penalty value on changes for the product width, thickness and hardness between pieces continuously rolled in the hot rolling plan [1]–[3]. Pan *et al.* [4] considered multi-objective hot-rolling scheduling problems contain minimizing virtual sheet-strips and change times of the thickness at same time maximal changes in thickness between adjacent sheet-strips. Pan and Yang [5] took into account order delivery time, production capacity besides penalty values and proposed a variant of column generation algorithm to solve the problem. Witt and Voß [6] established a nonlinear optimization model for the HRS problem with minimizing makespan as optimization objective, and designed a parallel genetic algorithm to solve the problem. Lyu *et al.* [7] studied continuous casting and HRS problems, established a mixed integer programming model and proposed a heuristic algorithm to solve it. Zhao *et al.* [8]–[10] consider hot rolling scheduling problems for wire rod and bar products and propose two-stage decomposition method [8], iterated greedy algorithms [9] and memetic algorithms [10] to handle them. Zhang *et al.* [11] and Chen *et al.* [12] both studied hot

The associate editor coordinating the review of this manuscript and approving it for publication was Hisao Ishibuchi.

rolling batch scheduling problems. Zhang *et al.* determine a sequence of the sheet strips with the objective of minimizing average thickness change and proposed a hybrid variable neighborhood search algorithm to solved the problem. Chen *et al.* decomposed the problem into a two-stage problem to solve. In the above researches, shuffling cost of slabs is not taken into account when slabs are charged into the furnace.

The operation efficiency of slab yard has a great impact on the production cost of hot rolling process. It is necessary to optimize the operation management that through hot rolling scheduling to reduce shuffling cost of slabs. In recent researches, some scholars begin to study HRS optimization considering slab stack shuffling (SSS). Tang *et al.* [13] established a nonlinear integer programming model for an HRS problem considering SSS and proposed a genetic algorithm. They assumed an ideal case that there is no intersection among candidate slab sets. Sing *et al.* [14] put forward an improved parallel genetic algorithm on the same problem. Wang *et al.* [15] proposed a two-stage heuristic for an HRS problem with SSS. Ren and Tang [16] established a nonlinear integer programming model for HRS problems with SSS considering crane capacity and converted the nonlinear model into a linear one. Different from existing studies, in this paper we study HRS problems considering SSS where there are intersections between candidate slab sets, and it is no need to put the shuffled slab back in place. To solve the problem, we establish an optimization model and propose a very large-scale neighborhood (VLSN) search algorithm.

The VLSN search algorithm was first proposed by Shaw *et al.* [17]. As a kind of metaheuristic, the VLSN search algorithm improves a solution by interactively destroying and repairing current feasible solutions. The VLSN search algorithm has been successfully applied to multiple combinational optimization problems. Breunig *et al.* [18] and Chen *et al.* [19] used the VLSN search algorithm to solve VRP. Sinclair *et al.* [20] implemented the VLSN search algorithm for an integrated aircraft and passenger recovery problem. Yu *et al.* [21] and Yagiura *et al.* [22] used this algorithm to solve assignment problems. Majid *et al.* [23] used this algorithm to solve supply chain network design problems. Brueggemann and Hurink [24] used this algorithm to solve a parallel machine scheduling problem. Guo *et al.* [25] and Fanjul-Peyro and Ruiz [26] used this algorithm to solve parallel machines scheduling problem. M. Zenker *et al.* [27] and Guo *et al.* [28] applied the VLSN search algorithm to railroad container terminals. In this paper, we apply the VLSN search algorithm to solve an HRS problem.

The main contributions of this paper include the following two parts: 1) We propose a special index method of slabs. By using it, the number of shuffles for selected slabs can be minimized by indirectly minimizing the sum of their serial numbers. In this way a hard procedure for calculating the number of shuffles can be avoided. 2) Based on the above idea, we establish a simplified integer linear programming (ILP) model that can approximately solve an HRS problem. This model can be optimally solved very fast by solving its linear relaxation since its constraint matrix is a totally unimodular matrix. 3) Taking the solution obtained by ILP model as an initial one, we propose a VLSN search algorithm to improve it in which a branch and bound (BAB) procedure is applied as a repairing process. We conduct extensive experiments to evaluate the proposed algorithm. The results show that the algorithm has great performance. Its high accuracy and fast speed imply its good potential to be used in practice.

The remaining part of this paper is organized as follows. Section 2 provides notations and an integer linear programming (ILP) model of the HRS problem. In Section 3, we propose a local search (LS) algorithm and a VLSN search algorithm. Section 4 presents the computational experiment results. In Section5, we summarize the research.

II. PROBLEM DESCRIPTION AND MATHEMATICAL MODEL

A. PROBLEM DESCRIPTION

In a practical production process, there are four linkage modes between steelmaking-continuous casting process and hot rolling process, which are continuous casting - cold charge rolling (CC-CCR) mode, continuous casting - hot charge rolling (CC-HCR) mode, continuous casting - direct hot charge rolling (CC-DHCR) mode, and continuous casting - hot direct rolling (CC-HDR) mode [29]. Except for CC-CCR linkage mode, in the other three linkage modes the slabs do not need to shuffling operation. Therefore, in this paper, we consider CC-CCR linkage mode only, means that all slabs in an HRS problem come from a slab yard.

An HRS problem consists of N ordered rolling slots. A slot i ($i = 1, \ldots, \mathrm{N}$) corresponds to a set C_i of candidate slabs from which we can select one slab for slot i. Slabs at the same rolling slot are similar in physical properties and order delivery date. Note that $C_l \cap C_k \neq \emptyset, 1 \leq \exists l, k \leq N, l \neq k$. An HRS problem is to select a slab for each slot i from C_i so as to minimize the number of shuffled slabs during the execution of hot rolling schedule. For an HRS problem including N hot rolling slots, the number of feasible solutions is about $\prod_{i=1}^{\mathrm{N}} |C_i|$, which grows exponentially with size of problems.

Slabs pile on stacks whose locations are fixed in a slab yard. Let s ($s = 1, \cdots, \mathrm{S}$) denote a slab, j ($j = 1, \cdots, \mathrm{P}$) denote a stack, and N_s be the set of slots in which slab s can be applied. The number of slabs piled on stack j is called height H_j of stack j. We number the slabs as following rules. The serial numbers of the top-level slabs are from 1 to P, then the serial numbers of the second top level slabs are from $\mathrm{P}+1$ to 2P, and so on. The level of slabs in stack$j$ counts from the top to the bottom, so that the level of top slab is 1, and the level of bottom slab is H_j. Let$s = f\ (j, h)$ be slab number whose level is h in stack j.

During execution of hot rolling schedule, selected slabs are charged into reheating furnaces according to the rolled order to be heated to a specific temperature. Only the top slabs in stacks are allowed to be charged. If a target slab is

not at the top of stacks, all the blocking slabs above it must be moved to a buffer area. Removing a slab into the buffer is called one shuffle. Shuffling operation is unproductive and occupy production resources. Shuffling operation not only increases crane workloads, but also increases the time of hot rolling processes such that is a bottleneck of the process. So, minimizing the number of the shuffled slab is taken as the optimization objective when we solve the HRS problems. The slabs in the buffer area will not be moved back to their original stacks and can be directly charged into heating furnaces without shuffles.

Example: We consider a small instance with N = 8 hot rolling slots, P = 6 stacks, and S = 37 slabs, as shown in Fig. 1. In Fig. 1, each box represents a slab. Grey boxes indicate candidate slabs. White boxes indicate slabs that are not involved in the schedule but may be shuffled. Slash boxes indicate slabs that are not involved in the schedule and cannot affect execution of the schedule. For example, $C_1 = \{1, 9, 19\}$ is the set of candidate slabs for hot rolling slot 1. $N_9 = \{1, 7\}$ is the set of slots which slab 9 can be applied in. The height of stack 1 is $H_1 = 4$.

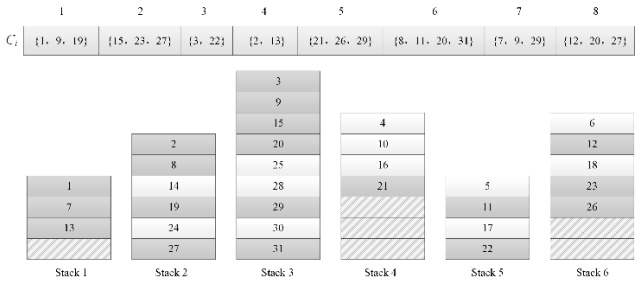

FIGURE 1. An instance of an HRS problem.

Table 1 presents a feasible solution of the instance. In Table 1, the row label "slot" represents hot rolling slot number, "selected slab" represents the selected slab number, and "shuffles" represents the number of shuffles for selected slabs. In this solution, we select slab $f\ (1, 1) = 1$ for slot 1, and it can be charged directly without shuffle. We select slab $f\ (3, 3) = 15$ for slot 2. To charge slab 15, it needs to 2 shuffles that slabs 3 and 9 must be moved to the buffer area firstly. We select slab $f\ (3, 1) = 3$ for slot 3, and it do not need shuffle anymore since slab 3 has been moved to the buffer area. By analogy, we calculate the number of shuffles for the remaining selected slabs. The total number of shuffles for executing the schedule is $2 + 4 + 5 = 11$.

TABLE 1. A feasible solution X_1 for the instance.

slot	1	2	3	4	5	6	7	8
selected slab	1	15	3	2	26	8	7	27
shuffles	0	2	0	0	4	0	0	5

B. MATHEMATICAL MODEL

We have established a mathematical programming model for the problem. The parameters and decision variables are defined as follows.

1) PARAMETERS

D_s The number of slabs upon slab s in original slab yard

$\varphi(s)$ stack that slab s is in

2) DECISION VARIABLES

$$x_{is} = \begin{cases} 1, & \text{if slab } s \text{ is selected for slot } i \\ 0, & \text{otherwise} \end{cases}$$

3) MATHEMATICAL MODEL

$$\min \sum_{i=1}^{N} \sum_{s \in C_i} N_{is} \times x_{is} \tag{1}$$

$$\sum_{i \in N_s} x_{is} \le 1, \quad s = 1, \cdots, S \tag{2}$$

$$\sum_{s \in C_i} x_{is} = 1, \quad i = 1, \cdots, N \tag{3}$$

$$N_{is} = D_s - \sum_{k=1}^{i-1} \sum_{n \in \{m \mid \varphi(m) = \varphi(s)\}} \min\{1, \max\{D_n - D_k, 0\}\} x_{kn},$$
$$i = 1, \cdots, N, \quad s = 1, \cdots, S \tag{4}$$

$$x_{is} \in \{0, 1\}, \quad i = 1, \cdots, N, \ s = 1, \cdots, S \tag{5}$$

$$N_{is} \in Z, \quad i = 1, \cdots, N, \ s = 1, \cdots, S \tag{6}$$

where, the objective function (1) minimizes the number of shuffles. Constraints (2) guarantee that each slab is assigned to at most a hot rolling slot. Constraints (3) guarantee that each hot rolling slot selects exactly one slab from its candidate slab set. Constraints (5) denote that if slab s is selected for the ith hot rolling slot, the number of shuffles is dependent on the selected slabs for the previous $(i-1)$ hot rolling slot which is in the same stack with slab s [13]. Constraints (4)-(6) provide value domain of variables. The model is complex and hard to solve.

C. THE ILP MODEL FOR APPROXIMATION OPTIMIZATION

According to the rule which we number slabs, the number for involved slabs in HRS problems satisfy that $\max_{1 \le j \le p} \{f\ (j, h)\} < \min_{1 \le j \le p} \{f\ (j, h+1)\}$, $h = 1, 2, \ldots, \max \left\{ \left(H_j - 1\right) | 1 \le j \le P \right\}$, as shown in Fig. 1. Basing on the index method of slabs, it can be seen that the closer slabs are to the top of stacks, the smaller slab numbers are, and the less slabs shuffled will be. Therefore, the number of shuffles for selected slabs is indirectly reduced by minimizing the sum of serial numbers for selected slabs. Based on this observation, we propose the following ILP model for approximatively

solving the HRS problem, which can effectively avoid the complex shuffling constraints.

$$\text{ILP:} \quad \min \sum_{i=1}^{N} \sum_{s \in C_i} s \times x_{is} \tag{7}$$

$$\text{s.t.} \sum_{i \in N_s} x_{is} \leq 1, \quad s = 1, \cdots, \text{S} \tag{8}$$

$$\sum_{s \in C_i} x_{is} = 1, \quad i = 1, \cdots, \text{N} \tag{9}$$

$$x_{is} \in \{0, 1\}, \quad i = 1, \cdots, \text{N}, \ s = 1, \cdots, \text{S} \tag{10}$$

where, the objective function (1) minimizes the sum of serial numbers for selected slabs. Constraints (8)-(9) are same with constraints (2)-(3).

The ILP model is a classic assignment problem whose constraint matrix is a totally unimodular matrix. So, optimal solutions of ILP model are the same as optimal solutions for linear relaxation of ILP model. Therefore, ILP model is easy to solve and can be used to quickly generate an approximate optimal solution. Table 2 presents a solution X_2 for the instance in Fig. 1 through solving ILP model with CPLEX. The objective value of solution X_2 is 6.

TABLE 2. Solution X_2 of ILP model for the instance.

slot	1	2	3	4	5	6	7	8
selected slab	1	15	3	2	21	8	7	12
shuffles	0	2	0	0	3	0	0	1

III. HEURISTIC ALGORITHMS

In this section, we propose two heuristics for the HRS problem. One is a local search (LS) algorithm, the other is a VLSN search algorithm.

A. THE LS ALGORITHM

We use LS algorithm to solve the HRS problem, which is a common search method in engineering. In the LS process, let X denote a solution and X^{cur} denote a current solution of the HRS problem, $X[i]$ be the selected slab number for hot rolling slot i in X, and $\Omega(X) = \{s \,|\, s = X[i], i = 1, \cdots, N\}$. Let cost (X) be the objective value of solution X. Starting from the current solution X^{cur}, each operation creates a neighbor solution X by reselecting a candidate slab for hot rolling slot i from the set C_i. Compared cost (X^{cur}) and cost (X), if an improving solution X is found, the current solution X^{cur} is replaced by X. The search traverses candidate slab sets of all the hot rolling slots, stopping when none of the neighbors can improve the current solution. Fig. 2 shows a search operation for solution X_2 of the instance in Table 2. The procedure of the LS algorithm is as follows.

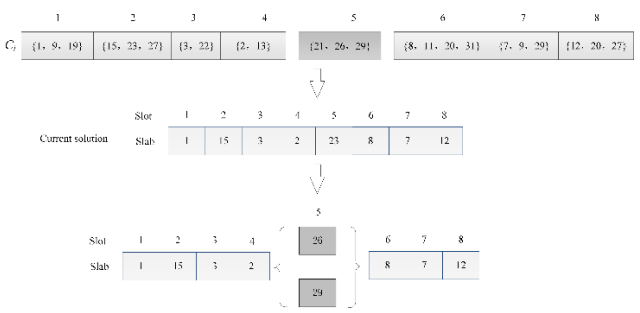

FIGURE 2. A search operation of LS.

Algorithm 1 The LS Procedure: LS (X)

Step 1: Get an initial solution X solving ILP model, $X^{cur} = X$, let flag *improved* $=$ *true*

Step 2: **while** *(improved)* **do**
- *improved* $=$ *false*;
- **For** $i = 1$ ***to*** N **do**
  - **For** $s \in C_i$ **do**
    - **If** $s \notin \Omega(X^{cur})$ **Then**
      - $X[i] := s$;
      - **If** cost $(X) <$ cost(X^{cur}) **Then**
        - $X^{cur}[i] := s,$;
        - *improved* $=$ *true*
      - **Else**
        - $X[i] := X^{cur}[i]$

Step 3: **Return** X^{cur}

B. THE VLSN SEARCH ALGORITHM

The VLSN search algorithm is based on the observation that searching a large neighborhood results in finding local optima of high quality. Starting from an initial feasible solution, the VLSN search algorithm iteratively improves the current solution by using destruction and reconstruction operations until the termination conditions are met. In the destruction operation, the algorithm releases some decision variable values in the current solution and retains the remaining variable assignments. In the reconstruction operation, the algorithm uses a specially designed procedure to reassign the remained variable values.

The proposed VLSN search algorithm starts from a feasible initial solution X. At every step, we release k hot rolling slots to reassign candidate slabs and the neighborhood is composed of all feasible assignments to released k hot rolling slots, while the selected slabs have been fixed for the remain $N-k$ hot rolling slots. Then we make use of a BAB algorithm to optimally reassign slabs to these released hot rolling slots forming a neighboring solution X^{new}. If the chosen neighboring solution has a lower cost of objective function than the current one, it becomes the new current solution. We set the number of iterations, stopping until exceeds the predefined limit. Fig. 3 shows a search operation for solution X_2 of instance in Table 2 and the neighborhood structure of the VLNS. Let sub_k denote the sub problem involving k released

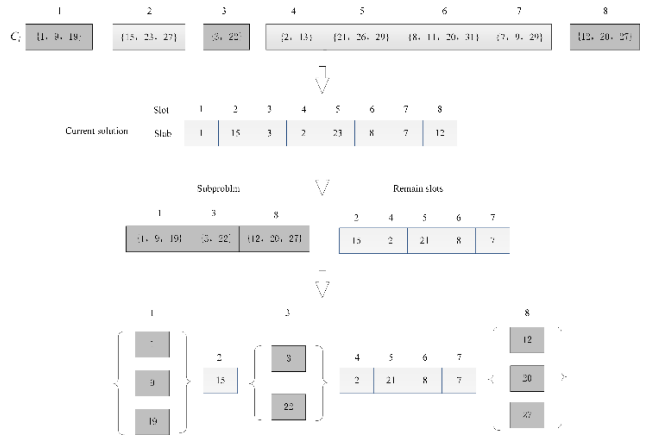

FIGURE 3. A search operation and neighborhood structure of VLNS.

hot rolling slots forming a set B_{sub_k}, and X_{sub_k} denote the optimal solution of sub_k. All other symbols have the same meaning as described above. The procedure of the VLSN search algorithm is as follows.

Algorithm 2 The VLSN Search Procedure: VLSN (X, k)

Step 1: Initialize parameter k,
$maxUnsuccess = N - k$, $count = 0$.
Get an initial solution$X$ using ILP model,
$X^{cur} = X$.

Step 2: **while** $(count \leq maxUnsuccess)$ **do**
a. Based on X^{cur}, release k hot rolling;
slot assignments to get a sub problem sub_k. ;
b. Solve sub_k by the BAB algorithm to;
get its optimal solution X_{sub_k}. ;
c. Combine X^{cur} and X_{sub_k} to get a;
new complete solution X^{new}.;
If $cost\,(X^{new}) < \text{cost}(X^{cur})$ **Then**
$X^{cur} := X^{new}$, $count = 0$
Else
$count{+}{+}$

Step 3: **Return** X^{cur}

In the following, we describe the destruction operators and reconstruction operators in details.

1) DESTRUCTION OPERATOR

For the HRS problem, destruction operator is to randomly release k hot rolling slots and reselect slabs for them. The operation is a key factor in the VLNS algorithm that the number of released hot rolling slots determines both the size of the neighborhood and the scale of subproblems.

2) RECONSTRUCTION OPERATOR

BAB algorithm is an exact method which is frequently used to solve combinatorial optimization problems [30], and we use it to solve a sub-problem sub_k. The BAB algorithm adopts depth-first search strategy and the maximum depth of the tree is k. During the solving procedure for sub_k, the objective

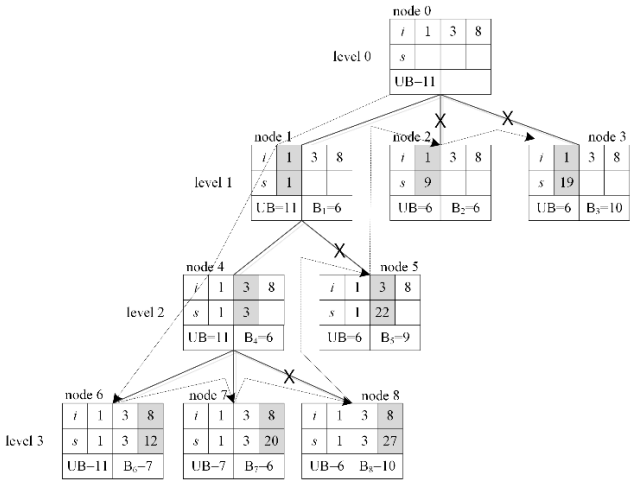

FIGURE 4. A BAB tree of solving a sub-problem.

TABLE 3. A new solution X_3 for the instance.

slot	1	2	3	4	5	6	7	8
selected slab	1	15	3	2	26	8	7	20
shuffles	0	2	0	0	4	0	0	0

TABLE 4. The parameters of test instances.

Set	N	P	$\lvert C_i \rvert$	H_j
1	50	40	{2,3,4}	{3, ...,11}
2	75	45	{2,3,4}	{3, ...,11}
3	100	60	{2,3,4}	{3, ...,11}
4	125	75	{2,3,4}	{3, ...,11}
5	150	90	{2,3,4}	{3, ...,11}
6	175	105	{2,3,4}	{3, ...,11}
7	200	120	{2,3,4}	{3, ...,11}

function value of nodes equals to the number of shuffles for reassigned slabs in the sub-problem and selected slabs in the main problem. Because if reassigned slabs for released hot rolling slots in sub-problem locate in same stack as selected slabs for retained hot rolling slots in the main problem, it will change the number of shuffles for the selected slabs. Since the current solution is feasible before the destruction operator is applied, we take the objective function value of the current solution as the upper bound (UB) of the BAB algorithm.

The BAB solution tree contains $k + 1$ levels. Level 0 represents the virtual root. Each level $l(l = 1, 2 \cdots, k)$ corresponds to a slot in sub_k and each node denotes a candidate slab. Based on the index method of slabs in section 2.1, we state that the smaller the selected slab serial number is, the less number of shuffles for slabs. Therefore, the branched nodes are sorted for each node in non-decreasing order according to serial number of slab that the tree will be pruned earlier in the BAB process.

There are two rules to prune branches in the search tree. The one, comparing with UB, we prune the branches in which

TABLE 5. Computational results of the VLNS search algorithm with different size sub-problems.

Set	VLNS-3		VLNS-4		VLNS-5		VLNS-6		VLNS-7	
	CPU	obj	CPU	obj	CPU	obj	CPU	obj	CPU	obj
1	12.15	53.0	14.08	52.9	14.53	51.9	18.69	51.8	41.71	51.6
2	13.33	75.2	19.23	73.9	26.16	73.5	30.11	73.1	44.26	73.0
3	32.08	89.3	41.3	88.9	50.39	88.7	62.36	88.1	81.00	87.7
4	49.89	115.4	77.7	113.4	87.55	113.1	127.06	112.6	175.31	111.2
5	69.15	140.3	82.13	139.9	99.99	138.8	185.73	138.7	309.54	138.3
6	77.37	157.7	92.38	156.5	110.82	155.0	372.61	154.9	537.73	153.3
7	90.77	182.5	103.53	180.3	120.30	179.0	458.17	178.2	848.76	177.8

TABLE 6. Comparison of the CH, ILP, LS and VLNS search algorithms.

Set	CH		ILP	LS			VLNS-5		
	CPU	obj	obj	CPU	obj	impr	CPU	obj	impr
1	19.73	95	64.6	0.29	50.1	22.45%	14.53	49.9	22.76%
2	31.34	133	94	0.7	73.7	21.60%	26.16	73.5	21.81%
3	46.79	192	111.8	1.39	92.5	17.26%	50.39	88.7	20.69%
4	70.98	257	143.8	2.64	116.5	18.98%	87.55	113.1	21.35%
5	89.35	294	172.9	4.04	145.35	15.93%	99.99	138.8	19.72%
6	117.52	349	193.2	6.17	161.1	16.61%	110.82	155	19.76%
7	145.67	389	221.7	8.74	189.5	14.52%	120.3	179	19.26%

the total number of shuffled slabs of assigned slots is bigger than UB. The other one is that branched slabs have been occupied by other slots, which means that we cannot find a feasible solution. For sub_k, a $k+1$-level tree can be formed containing $\prod_{i \in B_{sub_k}} |C_i|$ nodes at most by a BAB algorithm. In each operator, both the scale of selected subproblem and the capacity of the candidate slab set for each hot rolling slot are small, therefore, the BAB algorithm is efficient for solving subproblems.

Example: We randomly select hot rolling slot 1, 3 and 8 to form a sub-problem of the instance in Section 2.1. Fig. 4 shows a BAB tree of the sub-problem. The current solution is X_1 in Table 1, namely UB $= 11$. In Fig. 2 each node $h(h = 1, 2, \cdots)$ shows a branching status, the currentUB and the cost of the node B_h, where "i" represents slot number and "s" represents slab number. There are 4 levels in the tree, where level 0 is virtual root as node 0 and the other three levels correspond to hot rolling slot 1, 3 and 8 in turn. The BAB algorithm starts from node 0, which is branched into three nodes, namely node 1, node 2 and node 3, each of which denotes a candidate slab for hot rolling slot 1 in level 1. According to selection strategy, we selected node 1 denoting the smallest number slab in set C_1. By depth-first search strategy, since $B_1 <$ UB, node 1 can further be branched into two nodes, namely node 4 and node 5. Since node 4 contains a smaller number slab in set C_3 and $B_4 <$ UB, node 4 is branched into node 6, node 7 and node 8 in turn, all of which are leaf, and the corresponding objective function values are $B_6 = 7$, $B_7 = 6$ and $B_8 = 10$. The node 7 has the minimal objective function value $B_7 = 6$, so we update UB $= B_7 = 6$ and prune node 8. We traverse all the nodes forming the path $0-1-5$, $0-2$ and $0-3$ in turn by backtrack method, all which are pruned since $B_5, B_2, B_3 \geq$ UB. As shown in Fig. 2, dotted line indicates the whole depth-first search procedure. Since the minimum objective value is $6(cost\ (X_1) > 6)$ in the sub -problem, we can conclude that slab 1, 3, 20 are respectively selected for slot 1, 3, 8 in sub-problem to repair X_1forming a new solution X_3 as shown in Table 3 and the objective value is 6.

IV. COMPUTATIONAL RESULTS

In this section, we report our computational results of ILP model, the LS algorithm and the VLSN search algorithm. All the computational experiments were conducted on a desktop computer with 4 GB of RAM, the Windows 10 Professional 64-bit operating system, and an Intel Core i5 processor with four 3.3 GHz cores. Algorithms are implemented in C++ using CPLEX 12.61 and compiled with the Visual Studio 2013 C++ compiler. For all instances, the run time limit is set to 3,600 seconds for CPLEX solver.

A. INSTANCE GENERATION

In order to test the performance of the proposed model and algorithms, we generated 7 sets of instances simulating practical situations, each of which includes 10 instances. The HRS problems vary by the following parameters: the number of hot rolling slots(N), the number of stacks (P), the height of stacks ($H_j, j = 1, \ldots, P$), and the capacity of candidate slab sets($|C_i|, i = 1, \ldots, N$). In our experiments, the parameters of each instance set are shown in Table 6.

In each instance set, there are three sizes of candidate slab sets, whose capacities are 2, 3, and 4 slabs, respectively. The quantitative proportion for three sizes of sets is 2:3:2. Each stack randomly stores 3-11 slabs and candidate slabs are randomly distributed in the stacks. The number of candidate slabs accounts for about 2/3 of the total in the yard.

B. COMPUTATIONAL RESULTS

We solve each instance using VLSN search algorithms with the different size of sub-problems (VLSN-k)$k \in \{3, 4, 5, 6, 7\}$, taking solutions obtained by the ILP model as initial solutions. The computational results are shown in Table 5. In Table 5, column "obj" represents the average objective function values on 10 instances of each set. Column "CPU" represents the average solution time in second on 10 instances of each set.

Based on the results shown in Table 5 we note that as sizes of sub-problems increase, the objective function values of each set are improved, but more time is taken. Balancing time and objective function values, it is more reasonable to use VLNS-5 to solve HRS problems.

To the best of our knowledge, no the same or similar problems have been considered by existing papers. To evaluate the effectiveness and accuracy of LS and VLNS algorithms, we have compared our algorithms with a constructive heuristic method that is currently used in real-world steel plants. In the constructive heuristic (CH), we randomly select a slab for each hot rolling slot from its candidate slab set forming a feasible schedule and calculate the number of shuffling operations. To avoid the randomness of the constructive heuristic, we run it 10000 times and select the best solution among the found ones as a competitor. The results of objective function value and running time are shown in column "CH" of Table 6.

Besides, Table 6 compares the performance of the LS and VLNS search algorithms. Both the LS and VLNS search algorithms take solutions obtained by the ILP model as initial solutions. We take the objective function value of the initial solutions as the benchmark, and the improved ratio of the LS algorithm and the VLSN-5 algorithm are shown in column "impr" of Table 6. The improved ratio is computed as follows.

$$\text{improved ratio} = \frac{\text{obj}_{\text{ILP}} - \text{obj}_{\text{LS(VLSN-k)}}}{\text{obj}_{\text{ILP}}}$$

The results in Table 6 show that compared to the constructive heuristic, both the model ILP, LS and VLNS algorithms have significant advantages in terms of time and objective function value. Besides, the results show that starting from the same initial solutions, the improvement of the VLSN search algorithm is better than the LS algorithm. But the runtimes of the VLSN search algorithms are longer than that of the LS algorithm. Balancing runtime and quality of solutions, the VLSN search algorithm significantly improves objective function values in acceptable runtime ranges. This shows the VLSN algorithm's accuracy and efficiency.

TABLE 7. Comparison between the LS and VLSN search algorithms for different initial solutions.

Set		Solution				
		1	2	3	4	5
1	init	62	63	67	74	87
	LS	52	50	53	55	57
	VLNS	47	48	48	47	48
2	init	82	89	97	105	117
	LS	69	70	75	78	81
	VLNS	66	66	67	67	66
3	init	121	135	165	198	246
	LS	108	109	118	125	130
	VLNS	103	104	103	104	105
4	init	160	245	268	303	355
	LS	136	150	157	169	181
	VLNS	129	131	132	133	135
5	init	162	244	291	387	453
	LS	132	133	145	163	177
	VLSN	124	125	127	126	128
6	init	209	238	321	435	523
	LS	177	196	194	223	244
	VLSN	167	170	168	168	171
7	init	242	251	349	463	573
	LS	195	203	214	231	252
	VLSN	183	181	186	184	187

In order to test the influence of different initial solutions on performances of the LS and VLSN search algorithms, we take randomly generated 5 feasible solutions as initial solutions for LS and VLSN-5 algorithm to solve a stance in each set. The computational results are shown in Table 7.

Table 7 shows that as the objective function values of initial solutions increase, the objective function values of the LS algorithm increase significantly, but that of VLNS search algorithm have little change. It can be concluded that, comparing with the LS algorithm, quality of initial solutions has little influence on performance of the VLNS search algorithm, namely, the VLNS search algorithm is more robust than the LS.

V. CONCLUSION

In summary, we study HRS problems considering SSS. We build an integer linear programming to compute approximation solutions. Besides we propose the VLSN search algorithm to solve the HRS problems, which can effectively improve the quality of solutions with different scale of problems in acceptable time. At the same time, the VLSN search algorithm is robust since that does not depend on the quality of initial solutions. So, the proposed VLSN search algorithm is effective in both theoretical and practical production for HRS problems. In the future, we tend to explore other intelligence algorithms, e.g. monarch butterfly optimization, earthworm optimization algorithm, elephant herding optimization and moth search algorithm to solve the problem to further improve the solution quality and computing time. Meanwhile, we will attempt to find a suitable combinational optimization problem to which the HRS can convert and use it to obtain optimal solution of the problem. Besides, we will tend to

design effective methods to obtain good lower bounds of the HRS.

REFERENCES

[1] L. X. Tang, J. Y. Liu, A. Y. Rong, and Z. H. Yang, "A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex," *Eur. J. Oper. Res.*, vol. 124, no. 2, pp. 267–282, Jul. 2000, doi: 10.1016/S0377-2217(99)00380-X.

[2] S. Jia, J. Zhu, G. Yang, J. Yi, and B. Du, "A decomposition-based hierarchical optimization algorithm for hot rolling batch scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 61, nos. 5–8, pp. 487–501, Jul. 2012, doi: 10.1007/s00170-011-3749-9.

[3] S. X. Liu, J. H. Song, and S. C. Zhou, "Model and algorithm for solving hot strip rolling batch planning problems," *Control. Theory Appl.*, vol. 24, no. 2, pp. 243–248, Feb. 2007, doi: 10.1360/aas-007-0072.

[4] Q.-K. Pan, L. Gao, and L. Wang, "A multi-objective hot-rolling scheduling problem in the compact strip production," *Appl. Math. Model.*, vol. 73, pp. 327–348, Sep. 2019, doi: 10.1016/j.apm.2019.04.006.

[5] C. Pan and G. K. Yang, "A method of solving a large-scale rolling batch scheduling problem in steel production using a variant of column generation," *Comput. Ind. Eng.*, vol. 56, no. 1, pp. 165–178, Feb. 2009, doi: 10.1016/j.cie.2008.05.001.

[6] A. Witt and S. Voß, "Simple heuristics for scheduling with limited intermediate storage," *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2293–2309, Aug. 2007, doi: 10.1016/j.cor.2005.09.004.

[7] M. Lyu, Z. Wang, and F. T. S. Chan, "Mixed integer programming model and heuristic algorithm for production planning of continuous casting and hot rolling," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Gothenburg, Sweden, Aug. 2015, pp. 1503–1508, doi: 10.1109/CoASE.2015.7294312.

[8] Z. Zhao, S. Liu, M. Zhou, X. Guo, and L. Qi, "Decomposition method for new single-machine scheduling problems from steel production systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 3, pp. 1376–1387, Jul. 2020, doi: 10.1109/TASE.2019.2953669.

[9] Z. Zhao, S. Liu, M. Zhou, D. You, and X. Guo, "Heuristic scheduling of batch production processes based on Petri nets and iterated greedy algorithms," *IEEE Trans. Autom. Sci. Eng.*, early access, Oct. 23, 2020, doi: 10.1109/TASE.2020.3027532.

[10] Z. Zhao, S. Liu, M. Zhou, and A. Abusorrah, "Dual-objective mixed integer linear program and memetic algorithm for an industrial group scheduling problem," *IEEE/CAA J. Automatica Sinica*, early access, Dec. 29, 2020, doi: 10.1109/JAS.2020.1003539.

[11] B. Zhang, Q.-K. Pan, L. Gao, X.-L. Zhang, and Q.-D. Chen, "A hybrid variable neighborhood search algorithm for the hot rolling batch scheduling problem in compact strip production," *Comput. Ind. Eng.*, vol. 116, pp. 22–36, Feb. 2018, doi: 10.1016/j.cie.2017.12.013.

[12] Q. Chen, Q. Pan, B. Zhang, J. Ding, and J. Li, "Effective hot rolling batch scheduling algorithms in compact strip production," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 4, pp. 1933–1951, Oct. 2019, doi: 10.1109/TASE.2019.2914925.

[13] L. Tang, J. Liu, A. Rong, and Z. Yang, "Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules," *Int. J. Prod. Res.*, vol. 40, no. 7, pp. 1583–1595, Jan. 2002, doi: 10.1080/00207540110110118424.

[14] K. A. Singh, Srinivas, and M. K. Tiwari, "Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved parallel genetic algorithms," *Int. J. Prod. Econ.*, vol. 91, no. 2, pp. 135–147, Sep. 2004, doi: 10.1016/j.ijpe.2003.07.005.

[15] M. Wang, T. K. Li, and B. L. Wang, "Local search algorithm for the overlapped turned-out slab pile problem," *Comput. Integr. Manuf*, vol. 16, no. 3, pp. 658–662, Mar. 2010. [Online]. Available: http://en.cnki.com.cn/Article_en/CJFDTotal-JSJJ201003029.htm

[16] H.-Z. Ren and L.-X. Tang, "Study on modelling and optimization method for the slab stack shuffling problem considering area crane capacity," *Acta Automatica Sinica*, vol. 36, no. 4, pp. 586–592, May 2010, doi: 10.3724/SP.J.1004.2010.00586.

[17] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming*, 4th ed., vol. 1520, M. Maher and J. Puget, Eds. Berlin, Germany: Springer, 1998, doi: 10.1007/3-540-49481-2_30.

[18] U. Breunig, V. Schmid, R. F. Hartl, and T. Vidal, "A large neighbourhood based heuristic for two-echelon routing problems," *Comput. Oper. Res.*, vol. 76, pp. 208–225, Dec. 2016, doi: 10.1016/j.cor.2016.06.014.

[19] S. Chen, R. Chen, G.-G. Wang, J. Gao, and A. K. Sangaiah, "An adaptive large neighborhood search heuristic for dynamic vehicle routing problems," *Comput. Electr. Eng.*, vol. 67, pp. 596–607, Apr. 2018, doi: 10.1016/j.compeleceng.2018.02.049.

[20] K. Sinclair, J.-F. Cordeau, and G. Laporte, "Improvements to a large neighborhood search heuristic for an integrated aircraft and passenger recovery problem," *Eur. J. Oper. Res.*, vol. 233, no. 1, pp. 234–245, Feb. 2014, doi: 10.1016/j.ejor.2013.08.034.

[21] C. Yu, D. Zhang, and H. Y. K. Lau, "An adaptive large neighborhood search heuristic for solving a robust gate assignment problem," *Expert Syst. Appl.*, vol. 84, pp. 143–154, Oct. 2017, doi: 10.1016/j.eswa.2017.04.050.

[22] M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover, "A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem," *Discrete Optim.*, vol. 1, no. 1, pp. 87–98, Jun. 2004, doi: 10.1016/j.disopt.2004.03.005.

[23] E. Majid, D. Pierre, and P. Olivier, "A large neighborhood search heuristic for supply chain network design," *Comput. Oper. Res.*, vol. 80, pp. 23–27, Apr. 2017, doi: 10.1016/j.cor.2016.11.012.

[24] T. Brueggemann and J. L. Hurink, "Matching based very large-scale neighborhoods for parallel machine scheduling," *J. Heuristics*, vol. 17, no. 6, pp. 637–658, Nov. 2010, doi: 10.1007/s10732-010-9149-8.

[25] P. Guo, F. Weidinger, and N. Boysen, "Parallel machine scheduling with job synchronization to enable efficient material flows in hub terminals," *Omega*, vol. 89, pp. 110–121, Dec. 2019, doi: 10.1016/j.omega.2018.10.003.

[26] L. Fanjul-Peyro and R. Ruiz, "Size-reduction heuristics for the unrelated parallel machines scheduling problem," *Comput. Oper. Res.*, vol. 38, no. 1, pp. 301–309, Jan. 2011, doi: 10.1016/j.cor.2010.05.005.

[27] M. Zenker and N. Boysen, "Dock sharing in cross-docking facilities of the postal service industry," *J. Oper. Res. Soc.*, vol. 69, no. 7, pp. 1061–1076, Jul. 2018.

[28] P. Guo, W. Cheng, Y. Wang, and N. Boysen, "Gantry crane scheduling in intermodal rail-road container terminals," *Int. J. Prod. Res.*, vol. 56, no. 16, pp. 5419–5436, Aug. 2018, doi: 10.1080/00207543.2018.1444812.

[29] L. X. Tang, J. Y. Liu, A. Y. Rong, and Z. H. Yang, "A review of planning and scheduling systems and methods for integrated steel production," *Eur. J. Oper. Res.*, vol. 133, no. 1, pp. 1–20, Aug. 2001, doi: 10.1016/S0377-2217(00)00240-X.

[30] J. Gmys, M. Mezmaz, N. Melab, and D. Tuyttens, "A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem," *Eur. J. Oper. Res.*, vol. 284, no. 3, pp. 814–833, Aug. 2020, doi: 10.1016/j.ejor.2020.01.039.

YARONG SHI received the B.S. degree in automation from the Hunan University of Technology, Zhuzhou, China, in 2014, and the M.S. degree in systems engineering from Northeastern University, Shenyang, China, in 2016. She is currently pursuing the Ph.D. degree in systems engineering with Northeastern University, Shenyang, China. Her research focuses on planning and scheduling of steel production, mathematical programming, and heuristics algorithm.

SHIXIN LIU (Member, IEEE) received the B.S. degree in mechanical engineering from Southwest Jiaotong University, Sichuan, China, in 1990, and the M.S. and Ph.D. degrees in systems engineering from Northeastern University, Shenyang, China, in 1993 and 2000, respectively. He is currently a Professor with the College of Information Science and Engineering, Northeastern University, Shenyang, China. His research interests are in intelligent manufacturing, industrial big data, intelligent decision-making, production planning and scheduling. He has over 100 publications including one book.