A Fast Heuristic Algorithm for Ladle Scheduling Based on Vehicle Routing Problem with Time Windows Model

Zheng WEI,1,2) Tan ZHU,1) Tianzhang HE1) and Shixin LIU1)*  
1) College of Information Science and Engineering, Northeastern University, Shenyang, 110819 China.  
2) National Laboratory for Information Science and Technology, Tsinghua University, Beijing, 100084 China.  
(Received on March 1, 2013; accepted on July 16, 2014)

In the process of steelmaking and continuous casting production, an optimized ladle schedule will greatly reduce energy consumption and improve production. The ladle scheduling problem can be modeled as vehicle routing problem with time windows (VRPTW) and extra constraints. The main extra constraint is that components of the ladle, at the right time, have to be repaired no later than serving certain number of heats. The objective of ladle scheduling problem is minimizing the number of serving ladles and reducing the waiting time between serving two adjacent heats. According to the serving process of ladles, a mathematical model is established to solve this specific problem. In this paper, a three-step heuristic algorithm with time complexity of $O(n^2)$ is proposed, which is based on characteristics of the model and some preliminary experiments. The algorithm has been tested by several practical instances from a steel plant in China. Comparing with the schedules used in actual production, the computational results show that our algorithm optimizes the ladle schedules and solves the problem in less than 1 second, which proves the algorithm’s efficiency.

KEY WORDS: steelmaking and continuous casting; ladle scheduling; VRPTW; heuristic algorithm.

1. Introduction

In steelmaking-continuous casting production processes, ladle scheduling is an important problem for a steel plant to improve production and reduce energy consumption. A ladle begins to serve the molten steel at a converter or an electric furnace and finishes serving at a continuous caster (see Fig. 1), which is also called a heat. A heat begins with a ladle being filled with molten steel by a converter or an electric furnace. After the refining process, the molten steel will be poured into a continuous caster before the heat ends. The time for slag tipping is included in the heat because the time is certain for all heats. For the same reason, the transmission time before and after a heat can also be added. The fixed beginning time and end time of these heats in the production schedule are arranged in advance. The minimum number of ladles has to be arranged to serve all of heats with certain time windows in the production schedule (see Fig. 2). Therefore, the problem is translated into a vehicle routing problem with time window (VRPTW). In the production schedule, each ladle works as a vehicle and each heat can be taken as a task with time windows. A path has to be arranged for each ladle to serve several heats.

However, there are some extra constraints in this problem. According to the whole production process, three kinds of ladle components must be repaired after serving different certain number of heats and before serving the next one. It takes various length of time for a staff to replace or repair them. In details, the sliding sand of a ladle must be repaired each time after the ladle has serviced a heat. The slider of a ladle must be repaired after the ladle continuously services...
at most 4 heats. The nozzle of a ladle must be repaired after the ladle continuously services at most 8 heats. Hereinafter, we call the above three repairing operations of a ladle as operation a, operation b, and operation c, respectively. Each component has to be decided to be repaired or not. A good strategy is greatly helpful to make full use of ladles and reduce the number of ladles. Therefore, ladle scheduling can be transformed into the vehicle routing problem with time window and adjustment time (VRPTW-AT), which is presented for the first time by Tan et al. During the whole production process, reducing the number of ladles will help the steel plant cut down the cost. Besides, when a ladle is waiting to serve the next heat, it needs to be heated with burning gas in order to keep warm (see Fig. 1), which may waste plenty of gas. Under this circumstance, the objective is serving all these heats with the minimum number of ladles and minimal total time which all the heats and intervals occupy the ladles.

Vehicle routing problem with time window (VRPTW) has been researched in-depth. A variety of theories and algorithms were presented. For example, taboo search algorithm, ant colony algorithm and scatter search algorithm for VRPTW are discussed. However, they do not mention the repair requirement between two customers or tasks. In addition, most of the studies concerning ladles focus on improving material for heat preservation to prevent ladles from losing heat. To the best of our knowledge, only two papers proposed approaches for ladle scheduling problem. Wang et al. report a simulation system for ladle scheduling based on ladle matching priority rules. Tan et al. present the VRPTW-AT model for ladle scheduling for the first time. They proposed a scatter search algorithm to solve the problem successfully. Their experimental results show that the algorithm is efficient. However, there are some drawbacks in the algorithm. On one hand, they didn’t adjust repair time and the order of the component to be repaired positively. On the other hand, the time and space complexity of the algorithm are so high that it is hard to apply in steel plant and practical production.

In order to overcome these two drawbacks, we improve the mathematical model and design a fast heuristic algorithm based on characteristics of this kind of problem. Comparing with practical results and those of Tan’s algorithm for the same tapping schedule, our algorithm is more efficient. The following sections of this paper are organized as the followings. Problem description and mathematical model are described in Section 2. Section 3 is devoted to description of the solution procedure and three-step algorithm for the problem. Experimental design, computational results and discussions are presented in Section 4. Finally, conclusions are given in Section 5.

2. Problem Description and Mathematical Model

2.1. Problem Description

According to the definition of heat in Section 1, each heat must be served by only one ladle from the beginning to the end. After a ladle finishes serving a heat, the empty ladle has to be moved to a mending workshop to repair the certain components. When the repair is finished, it will be heated to keep in high temperature. The heating process will not stop until the ladle begins to serve another heat again. This work flow of a ladle has been illustrated in Fig. 1. There are more details in this process but this part of information is enough for our mathematical model. The objective for ladle scheduling is to reduce the number of ladles used in the production process and the interval time between adjacent heats served by the same ladle. The less ladles mean the less cost. The less interval time means the less energy consumption.

In ladle scheduling problems, we need to arrange the path of each ladle and the categories of component to be repaired. In order to describe a solution of the problem, we define solution set $L = \{L_1, L_2, \ldots, L_m\}$ where $L_k$ is the path of ladle $k (k = 1, 2, \ldots, m)$. A path should record heats serving order and repairing operations order, thus we can define $L_k = \{l(1), R_k(1), l(2), R_k(2), \ldots, l(n_k-1), R_k(n_k), l(n_k)\}$, where $l(i)$ is the $i$th $l(i)$ heat in the path of ladle $k$, and $R_k(j) (R_k(j) \subseteq \{a, b, c\})$ is repairing operations set. The components are among three kinds of components repairing, which are selected to be repaired between heat $j$ and $j + 1$.

As shown in Fig. 3, the rectangle with $l(i)$ is a heat serial number and the rectangle with $R_k(j)$ and $a, b, c$ is the repair operation between two adjacent heats served by a certain ladle.

Except for the common constraints of VRPTW, there are some extra constraints in VRPTW-AT:

1. A type of ladle can serve several certain kinds of molten steel;
2.2. Mathematical Model

The mathematical model is based on Tan’s model. 1) We have improved it by simplifying the model and skipping the factors that may not affect the results of the problem. The model has the following parameters:

- \( N = \{1,2,\ldots, n\} \) The set of heats in production schedule, they are in ascending order of heats beginning time. (iii)
- \( M = \{1,2,\ldots, m\} \) The set of available ladles. (iii)
- \( i \) The serial number of heats, \( i = 1,2,\ldots, n \). (iii)
- \( k \) The serial number of ladle paths, \( k = 1,2,\ldots, m \). (iii)
- \( n_k \) The number of heats in ladle path \( k \). (ii)
- \( s_i \) The start time of heat \( i \). (ii)
- \( e_i \) The end time of heat \( i \). (ii)
- \( c_i \) Cast number of heat \( i \). (ii)
- \( f_i \) The serial number of a converter or an electric furnace corresponding to heats \( i \). (ii)
- \( t_{ic} \) Transmission time from caster \( c \) to tilting turntable \( v \). (iii)
- \( t_{iv} \) Transmission time from tilting turntable \( v \) to converter or electric furnace \( f \). (iii)
- \( B_i \) Repair time of ladle \( k \) after serving heat \( i \). (ii)
- \( p_a \) Operation time of adding sliding sand. (iii)
- \( p_b \) Operation time of replacing a slide plate. (iii)
- \( p_c \) Operation time of replacing a sliding gate nozzle. (iii)
- \( d_a \) The maximum number of heats continuously serviced between repairing operations of blowing sand in one ladle path. (iii)
- \( d_b \) The maximum number of heats continuously serviced between repairing operations of replacing slide plate in one ladle path. (iii)
- \( d_c \) The maximum number of heats continuously serviced between repairing operations of replacing nozzle in one ladle path. (iii)
- \( x_{ijk} \) If heat \( i \) is the \( jth \) heat in ladle path \( k \), \( x_{ijk} = 1 \), else \( x_{ijk} = 0 \). (i)
- \( y_{ik}^a \) If the operation of blowing sand is performed in ladle path \( k \) after it has serviced heats \( j \), \( y_{ik}^a = 1 \), else \( y_{ik}^a = 0 \). (i)
- \( y_{ik}^b \) If the operation of replacing slide plate is performed in ladle path \( k \) after it has serviced heat \( j \), \( y_{ik}^b = 1 \), else \( y_{ik}^b = 0 \). (i)
- \( y_{ik}^c \) If the operation of replacing nozzle is performed in path \( k \) after it has serviced heat \( j \), \( y_{ik}^c = 1 \), else \( y_{ik}^c = 0 \). (i)

Hence, we could establish following mathematical model for ladle scheduling.

\[
\begin{align*}
\text{Min} & \quad \sum_{i=1}^{n} \sum_{k=1}^{m} x_{ijk} \quad \text{.................................. (1)} \\
\text{Min} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n_i-1} s_{ij} - t_{ij} \quad \text{.................................. (2)} \\
\text{s.t.} & \quad \sum_{k=1}^{m} x_{ijk} = 1, j = 1,2,\ldots, n \quad \text{............... (3)} \\
\sum_{i=1}^{n} x_{ijk} = 1, k = 1,2,\ldots, m, j = 1,2,\ldots, n_k-1 \quad \text{............... (4)} \\
\sum_{i=1}^{n} x_{ijk} \left(s_i - t_{ijk}^2\right) - \sum_{i=1}^{n} x_{ijk} \left(e_i + t_{ijk}^c\right) \geq B_j, \quad \text{............... (5)} \\
\sum_{j=1}^{u} y_{ik}^a \geq 1, u = 1,2,\ldots, n_k - d_a, k = 1,2,\ldots, m \quad \text{............... (6)} \\
\sum_{j=1}^{u} y_{ik}^b \geq 1, u = 1,2,\ldots, n_k - d_b, k = 1,2,\ldots, m \quad \text{............... (7)} \\
\sum_{j=1}^{u} y_{ik}^c \geq 1, u = 1,2,\ldots, n_k - d_c, k = 1,2,\ldots, m \quad \text{............... (8)} \\
x_{ijk} \in \{0,1\}, i = 1,2,\ldots, n, j = 1,2,\ldots, n_k \quad \text{............... (9)} \\
y_{ik}^a \in \{0,1\}, y_{ik}^b \in \{0,1\}, y_{ik}^c \in \{0,1\}, i = 1,2,\ldots, n_k \quad \text{............... (10)} \\
\end{align*}
\]

Where, objective function (1) expresses minimum number of ladles used; objective function (2) expresses minimum total interval time between every two adjacent heats in the premise of function (1); constraints (3) express one heat could just be serviced by only one ladle; constraints (4) express one ladle could just service one heat at the same time; constraints (5) express the interval between two consecutive heat is greater than or equal to the transit time between the equipment and repairing time of the ladle; constraints (6)–(8) express a ladle should be repaired in a specific way (adding sliding sand, changing slide plate and changing nozzle) after reaching the maximum times of serving consecutive furnaces; And, constraints (9) and (10) express the range of the decision variables.

2.3. The Characteristic of the Problem

According to the objective of the problem, minimizing the number of used ladles is the most essential goal. In order to achieve the goal, we should reduce time conflicts among heats at the same time. During the whole process, the maximum number of heats conflicts at the same time is the minimum required number of ladles. It can be fulfilled by scating heats in all ladle paths appropriately. It also means to expand interval between adjacent heats and reduce variance among them. However, the second goal is to reduce

*1: Decision Variable; ii: Subordinate Variable; iii: Constants.
length of intervals. We can calculate total interval \( I \) by 
\[
I = \sum_{i=1}^{n} \left( I_{\min} - s_i - e_i \right)
\]
based on expression (2), where all variables refer to Section 2.1 and 2.2. In the expression, we can know that value of \( I_{\min} \) is stable and hence conclude \( I_{\min} = \sum_{i=1}^{n} s_i - \max \sum_{i=1}^{n} s_i + \min \sum_{i=1}^{n} e_i \), which requires ladles to begin serving as late as possible and stop serving as early as possible. In Fig. 4, all of these phenomena are presented. Via a series of optimization methods, ladle schedule on the left will be transform into the one on the right with less ladles and shorter interval time.

In order to solve this contradictory, we have to formulate various strategies in different steps to overcome it. Inspired by the push-forward insertion heuristic, we put forward a new strategy and complete different heuristic functions to insert all heats in three processes to solve this problem.

3. The Three-step Heuristic Algorithm

3.1. The Data Structure

Inserting and reinserting are main methods of the algorithm. In order to operate the data more efficiently, we have designed the following data structure. In the ladle path array, each record includes a ladle’s available status flag and a path pointer. The available status flag records whether the ladle stops serving or not. In a ladle path, each record includes a member to describe the served heat number (line 1 in Fig. 5), 3 members to respectively describe whether the certain component of the ladle has been repaired before serving this heat (line 2–4 in Fig. 5) and 3 members to respectively describe the number of heats that have been served by the ladle after the certain component has been repaired (line 5–7 in Fig. 5). By using this structure, it is more convenient to calculate the relevant values. And, it is also more efficient to search, add and delete a heat. Figure 5 is an example of data structure. All components in the figure as shown have just been mentioned. The length of each array is changeable because we can not decide it in advance.

3.2. The Forward-insert Process

First of all, we need to sort the heats in ascending order according to their beginning time. Then we can begin to insert all heats one by one with the following strategy:

\[
\text{curHeat} = \text{lastHeat} + 1
\]

Where, \( \text{curHeat} \) means the serial number of a heat that is being inserted in this loop and \( \text{lastHeat} \) means the first heat’s serial number of the last added ladle. In the strategy, firstly, there is no ladle path and hence we need to add a ladle with its first heat (heat 0 in Fig. 6). Secondly, each heat may have multiple choices of ladle paths to join thus we need to evaluate the score of each choice. Finally we can select the best choice with the best score (minimum value) and add a heat to the path (see the process of adding heat 6 in Fig. 6). However, there may be no choice for the heat (the minimum value of score is positive infinity). In this case, another ladle has to be added (heat 1, 2 or 3 in Fig. 6). Besides, we should record the first heat of the last added ladle, which is useful in forward-reinsert process (section 3.4).

In order to achieve the goal, we design an evaluation function. When heat \( j \) is to be added, we assume that the ladle set is \( L = \{ L_1, L_2, \ldots, L_n \} \) and in each ladle path \( L_k = \)
been served by ladle end time and repair component type length(time)

\{l(1), R(1), l(2), R(2), \ldots, l(n_k - 1), R(n_k - 1), l(n_k)\}. If heat \( j \) adds into the ladle path \( i \), \( L_i = \{l(1), R(1), l(2), R(2), \ldots, l(n_k), R(n_k), l(n_k + 1)\} \). The score \( S_i \) can be calculated by

\[
S_i = \begin{cases} 
0 & \text{if } \left( s_j - \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \right) \vee \left( \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \right) \leq s_j \leq \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u , \\
+\infty & \text{if } \left( s_j - \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \right) > \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \\
\sum_{u \in \{a,b,c\}} \left( r_{(n_k)} u + \sum_{w_{(n_k)} u} \frac{P_u}{d_u} \right) & \text{if } \left( s_j - \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \right) > \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u , \\
+\infty & \text{if } \left( s_j - \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \right) > \epsilon_{(n_k)} + \sum_{u \in \{a,b,c\}} P_u \\
\end{cases}
\] 

(11a)

(11b)

(11c)

(11d)

where \( u \in \{a,b,c\}, W_{(n_k)} \) is the number of heats that have been served by ladle \( k \) after component \( u \) had been repaired last time. \( r_{(n_k)} \) can be decided by

\[
r_{(n_k)} = \begin{cases} 
1, w_{(n_k)} = d_u & u \in \{a,b,c\} \\
0, w_{(n_k)} \neq d_u 
\end{cases}
\]

(12)

In the first case (see expression 11a), the interval between the end time of the ladle path and the beginning time of heat \( j \) is longer than the last time for all components to be repaired. Moreover, end time of the path is earliest among all paths. If we use other ladles to serve this heat, it will waste more time during the interval. For example, in Fig. 7, we have two choices to add heat 6. In Fig. 7(a), heat 6 is added to \( L_i(i = 3, j = 6) \) while in Fig. 7(b), heat 6 is added to \( L_i(i = 4, j = 6) \). Apparently, the situation in Fig. 7(a) is better because when heat 7 has been added, we can find that Fig. 7(a) has an extra chance to repair component \( b \) in \( L_i \). In order to discard other choices, we define zero as its value.

In the next three cases (11b, 11c and 11d), we need to check whether end time of the last heat in a path is earlier than beginning time of the next adding heat. Besides, the components that have to be repaired will need extra time. Therefore, \( \epsilon_{(n_k)} + \sum P_u r_{(n_k)} \) and \( s_j \) should be compared.

In the second case (11b) and the third case (11c), \( \epsilon_{(n_k)} + \sum P_u r_{(n_k)} \leq s_j \) and hence path \( i \) is available for heat \( j \).

In the second case (see expression 11b), there are some extra constrains. If heat \( j \) has been added in path \( i \) and the next heat, heat \( j + 1 \), can create the interval \( (t_{\text{end}} - t_{\text{start}}) \) long enough for all components to be repaired after it is added to the path with earlier end time \( (t_{\text{end}}) \) compared to the end time of path \( i \) (exclude heat \( j \)). Under this circumstance, even though the value of \( \sum_{t_{\text{start}}}(\sum_{w_{(n_k)} u} \frac{P_u}{d_u} + \epsilon_{(n_k)}) \) may lower than the value has been calculated when heat \( j \) is added to other paths, it will also waste plenty of interval time by reducing chance for more components to be repaired. Therefore, this choice should be abandoned and its value is positive infinity. For example, in Fig. 8, we have two choices to add heat 5. In Fig. 8(a), heat 5 is added to \( L_i(i = 2, j = 5) \), heat 6 is added to \( L_3 \), while in Fig. 8(b), heat 5 is added to \( L_i(i = 3, j = 5) \) and heat 6 is added to \( L_2 \). Apparently, situation in Fig. 8(a) is better because after heat 6 is added, we can find that Fig. 8(a) has an extra chance to repair component \( b \) in \( L_3 \).

In the third case (see expression 11c), firstly, we have to decide \( R_{(n_k)} \subseteq \{a,b,c\} \) in following strategy.

1. (1) Choose components that have to be repaired, add them into \( R_{(n_k)} \);
2. (2) Calculate the interval still remained by \( I_{\text{left}} = s_j - \epsilon_{(n_k)} - \sum_{u \in R_{(n_k)}} P_u u \subseteq R_{(n_k)} \);
3. (3) Find all \( C \) which conform \( \{C \subseteq \{a,b,c\} - R_{(n_k)}\} \wedge \)
\(|\sum_i p_i < I_{s_i}, u \in C_0\); (4) Record \(v_0\) among \(v\) if \(\sum_i w_i p_i / d_i, u \in C_0\) is larger than any other \(C_i\). If there are two \(v\) with the same value, choose the one with larger \(\sum_i p_i, u \in C_i\). If \(\sum_i p_i\) of them are also the same, choose the set with components of longer repair time.

(5) \(R_t(n_i) = R_t(n_i) \cup C_i\).
After that, heat \(j\), as well as \(l(n_i + 1)\), and \(R_t(n_i)\) can be added to \(L_i\).

There are two parts in \(S_i\) of the third case. \(\sum_{i=1}^{m} \sum_i w_i p_i / d_i\) is the total repair time that the ladle hasn’t spent yet but it may have to spent later. So the later heat will have more chance to crash if the value is larger. \(l(n_i)\) is the total end time of all ladle path. The later heat will also have more chance to crash if the value is larger. The less crash means the less extra ladles to be used. So the choice should decrease its value as much as possible.

In the fourth case (see expression 11d), because \(e_{t(n_i)} + \sum_i p_i r_i > s_i\), which means there will be a crash if the heat is added to this path, and the heat can not be added into the path or there may be time conflicts. In order to abandon this choice, we define positive infinity as its score.

We can calculate \(S_i = \min\{S_i\}, i = 1, 2, \ldots, m\). Heat \(j\) can be decided to add to the ladle path \(k\) if \(S_i\) is not infinity.

Otherwise, a new ladle should be added. After each heat is distributed to one of the ladle path, the forward-insert process is completed. In this step, we have achieved two main goals of the task. One is to minimize the quantity of ladle. The other is to delay the ladle’s service start time. It is fulfilled by optimizing the heat serving order to make full use of the interval time to repair different types of components. The effect of forward-insert process can be seen in Fig. 9. Based on the formulation of the algorithm, it is hard to reduce any more ladles, because all ladles are added when there is no available path for a heat to insert. If we attempt to reduce more ladles, we must improve the expression (11).

### 3.3. The Backward-insert Process

The forward-insert process can greatly reduce the number of ladles and the total length of interval time in the front half of the ladle schedule. However, the heats will be quite loose in the later half (In Fig. 9, the heats arrangement hasn’t been compressed). Therefore the next goal is to arrange for the ladle to stop serving as early as possible. The main contribution of this process is to search all chances that a ladle might stop serving remainder heats.

First of all, we need to sort the heats in descending order according to the end time but we still use the heat serial number in forward-insert process. For convenience, we describe the heat number by \(h(i), i = 0, 1, \ldots, m - 1\) where \(i\) is the serial number in the descending order of the end time. In this process, each heat will be added to the path with following algorithm.

Fig. 8. The superiority of the strategy in the second case.

Fig. 9. The comparison between before and after the optimization.

**Ladle**

<table>
<thead>
<tr>
<th>Time</th>
<th>Optimized</th>
<th>Difference of time-consuming</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 10. An example of backward-insert process.
When heat $h(i)$ is to be added, the algorithm should attempt to search for the path with the latest start time. If the path is found, heat $h(i)$ will be added to this path (see the process of adding heat $h(4)$ in Fig. 10). If there is no path available, adding a new ladle with this heat and pushing the heat into the heat stack (see the process of adding heat $h(3)$ in the first half of Fig. 10). In this way, we can know how many ladles are necessary before the time of the end heat via seeing the top the two stacks in the Section 3.4 (more details of the stacks will be described in Section 3.4). For example, in Fig. 10, 4 ladles are necessary before time: 15016. In this process, we don’t need to concern about component repair because we just try to find the earliest possible time to stop each ladle to serve remaining heats. In other words, if ladle $i$ with the latest heat $h(j)$, the $i$th ladle can not be stopped earlier than the end time of $h(j)$. We can prove it by using math induction and reduction to absurdity easily.

**Proof:**

(1) When $i = 1$, $h(j)$ is the first heat added to ladle path 1. Apparently, path 1 can not stop earlier than the end time of $h(1)$ because it is the last heat in continuous steelmaking process. Therefore, the proposition is established.

(2) Assume that when $i = 1, 2, \ldots, k$, the proposition is established and when $i = k + 1$ the statement is false, which means path $i = k + 1$ with last heat $h(j_0)$ can be stopped earlier. Then $\exists j$ conforms $t_{h(i)} < t_{h(j)}$, heat $h(j)$ can take place $h(j_0)$ as the last ladle of path $i$. However, in the algorithm, $h(j_0)$ has to be arranged before $h(j)$ and $h(j_0)$ can’t join any other paths without clash when $h(j_0)$ is to be added. We can infer that if $h(j)$ is the last of ladle of path $i$ then $h(j_0)$ will crash one of the heats. Therefore, when $i = k + 1$, the proposition is also established.

Based on (1) and (2) we can proof this proposition.

**End proof.**

### 3.4. The Forward-reinsert Process

Based on the results of forward-insert and backward-insert, we can begin to optimize the later half. We will use all heats and the path data in the forward-insert process and the two stacks in the backward-insert process. This is the algorithm:

**Forward-reinsert algorithm**

- Firstly, all the heats added after the first heat of the last ladle should be removed (Fig. 11(a)). Then we can begin to reinsert the heats remained. Before a heat is to be added, the algorithm should attempt to stop a ladle’s serving when there is possibility for the plan to stop the path thus it is necessary to check the relationship between possible the least number of ladles and the number of current ladles.
The program gains this information via comparing the value of top element of the stack and the current condition. For example, in the second half of Fig. 11(a), current heat is 17, which is in the front of the heat stack top: 21 (or the top of the two stacks should be popped), and there are 7 ladles at present, which is more than 6 necessary ladles, at the top of the ladle stack. Therefore, it is possible to stop a ladle. If it is impossible, current heat should be added by strategy in Section 3.2. Otherwise, the program should do following process. A priority queue will save all the paths that can be stopped in ascending score (in the second half of Fig. 11(a)). We also design a function for score evaluating.

\[ S_k = \begin{cases} \epsilon^{(n_i)} - \sum_{u} \frac{w_u}{d_u}, & \text{ladle } k \text{ is available}, u \in \{a, b, c\} \\ + \infty, & \text{ladle } k \text{ is unavailable} \end{cases} \]

Where, all variables are similar to the ones presented in forward-insert process. The path with earlier end time and more time required for repairing components later will be more likely to be stopped. The goal is to stop the paths as early as possible so we should choose the path according to the end time. Besides, the path with more potential repair time may increase the crash chance of later heats so it is better to stop it rather than the one with less potential repair time. Therefore, the score evaluating function combines both of these two factors. Ladle \( k \) is selected by the algorithm when it conforms following condition:

\[ S_k = \min \{ S_j \}, \ j = 0, 1, \cdots, m \wedge \text{ladle } k \text{ is available} \]

After all the paths have been inserted in the priority queue according to the score, we should ensure the choice is available (the choice will not lead to unavoidable time conflicts among later heats that is to be inserted) thus a test should be done: all remaining heats will be inserted with the algorithm in Section 3.2 (see the first half of Fig. 11(b)). When we use the algorithm in Section 3.2 in this process, we should be concerned about two main differences. One is the heat should not be added to the stopped path. The other is the heat can only be added to the existing path and adding extra ladles is forbidden. When adding extra ladle is needed, it means a crash has happened. If there is a crash, this path-stop choice should be abandoned. We should recover the site before the test and then begin to test the next choice (see the first half of Fig. 11(b), delete all heats with dash edge including the gray one). Otherwise, we should recover the site except current heat (in Fig. 11(b), delete all heats with dashed edge excluding the gray one) and begin next loop for the next heat. When all the path-stop choices are unavailable (the priority queue is empty), the current heat has to be added by using the strategy in Section 3.2. All of these processes can be seen in Fig. 12. As Fig. 12 shown, when heat 0 to \( i \) have been decided, heat \( i + 1 \) is to be checked. There are 4 paths that can be stopped (\( L_1, L_2, L_5, L_6 \)) with ascending order according to the scores. In Fig. 12(a), there is a crash when \( L_2 \) is stopped and heat \( j_1 \) is added. Hence, heat \( i + 1 \) to \( j_1 \) should be cleared and enable \( L_2 \). And then \( L_5 \) is disabled. There is no clash of adding remaining heats. In this situation, we should clear heat \( i + 2 \) to \( j_2(n) \) and begin to check heat \( i + 2 \). In Fig. 12(b), all possible stopped paths will lead to clashes after the remaining heats have been added. Under this circumstance, none of extra paths should be disabled and add \( i + 1 \) with the strategy in Section 3.2.

3.5. Analysis of Time Complexity and Space Complexity

In the forward-insert process, the time for calculating the heuristic value (see expression 11) should be considered firstly. When \( S_i \) is computed, the third case spends more time than the other cases. Calculating \( \epsilon^{(n_i)} + \sum_{u} p_u f_u \leq s_j \) needs time \( O(1) \). Searching the subset of \( R(n_i) \) needs \( O(2^c) \), where \( c \) is a constant. Calculating \( \sum_{k=1}^{m} \left( \sum_{u} \frac{w_u}{d_u} + \epsilon^{(n_i)} \right) \) needs \( mO(c + 1) \). Taking the time complexity of quick sort into consideration, therefore, the total time complexity is \( nO(1) + O(2^c) + mO(c + 1) + O(n \log n) = O(mn + n \log n) \).

In the backward-insert process, the algorithm will spend \( O(m) \) to find the path for each heat to be added. There are \( n \) heats in total. So the time complexity is \( O(mn) \).

In the forward-reinsert process, the main strategy of inserting heats is similar to the first process but it needs to check the ladle which is available to be stopped. In this part, the worst situation is that all ladle path should be checked before a ladle is to be added, which will spend \( O(mn) \).
Besides if the result of the second process doesn’t work in this process, every heat needs to spend $O(mn)$. So the total time complexity is $O(mn^2)$.

Combining the three processes, we can calculate the total complexity at worst situation by $O(mn + n \log n) + O(mn) + O(mn^2) = O(mn^2)$. $m$ is the number of the ladle, which is related to the production ability. When $n$ is large enough (a long production schedule), compare to $\log n$, $m$ can be seen as constant. So, we can assert that the total complexity at worst situation is $O(n^2)$. According to the data structure and the algorithm, the space complexity is $O(n)$.

4. Experiments and Results

4.1. Experiments for Practical Data

In this section, we will apply our algorithm on three practical data sets to observe its optimal effect.

4.1.1. The Input of the Experiments

In this study, we test our algorithm on three practical instances obtained from a steel-plant in China. There are 2 basic oxygen furnaces, 2 LF refine furnaces, 2 continuous casters and 3 repair workshops in the factory. The tapping schedule S1, S2 and S3 consist of 49, 67 and 79 heats respectively, which is completely the same with Tan’s dataset. We also compare the results with those of scatter search algorithm designed by Tan. The algorithm is implemented in C++ language. The experiment was done on the computer with Intel Core i5 CPU, 4 GB interval storage and 64 bit Windows 7 operation system. The followings are the parameters setting that used in the study: $p_a = 2$ min, $p_b = 12$ min, $p_c = 18$ min, $d_a = 1$, $d_b = 5$, $d_c = 8$. Tan also choose these values in her experiment according to the steel plant.

4.1.2. The Experimental Results and Analysis

According to Table 1 and Fig. 13, we can easily find that the effect of 3-Step Algorithm is better than Scatter Search’s to solve this problem. It can be summarized in 3 aspects:

1. Optimization effect is better. Both algorithms can reduce the same ladles in the same tapping schedule but the 3-Step Algorithm can reduce more interval time.

2. The efficiency of algorithm is higher. SS needs to spend several minutes to solve the problem. However, 3-Step just needs several milliseconds.

3. In terms of stability, 3-Step is a nonrandom algorithm. So its result is changeless. SS is an algorithm based on random so its solution is random. But Tan didn’t give statistical analysis of the results so its stability is unsure.

4.2. Experiments for Simulation Data

Based on practical data sets and some optimization strategies$^5$ used in practical ladle scheduling, we get several simulation data sets from our simulation program. Our algorithm had also been applied to solve these instances. We can analyze the performance of 3-Step algorithm under different circumstances more comprehensively. The environment is the same as the environment in Section 4.1.

4.2.1. Simulation Data Generation

The various production processes and different types of molten steel will affect the length of heats. There is one or two production processes in each dataset thus we can easily distinguish the heat of each production process by its length. We classify them by production processes. In order to reduce the number of variable, one group is selected for simulation. We assume that the length of heats in this group normally distributed because the probability of heats arriving on time follows normal distribution. The average and standard deviation of the sample, with the range (4 000, 6 000), in this group is 4 653 and 379 respectively. Apparently, the scale of the part that can be optimized is different when densities of heats in a certain time are not the same. Hence, we should study them with different densities which can be decided by the length of interval between two adjacent heats served by

<table>
<thead>
<tr>
<th>plan</th>
<th>heat</th>
<th>algorithm</th>
<th>interval (min)</th>
<th>CPU (s)</th>
<th>ladle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49</td>
<td>3-Step</td>
<td>3851</td>
<td>0.015</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS</td>
<td>3851</td>
<td>57.23</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>practical</td>
<td>5039</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
<td>3-Step</td>
<td>8407</td>
<td>0.015</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS</td>
<td>8476</td>
<td>171.63</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>practical</td>
<td>8372</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>79</td>
<td>3-Step</td>
<td>9285</td>
<td>0.031</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS</td>
<td>9285</td>
<td>509.55</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>practical</td>
<td>11813</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1. The performance of SS and 3-Step algorithm.

Fig. 13. Statistics of the performance of 3-Step and Scatter Search algorithm.
one ladle. Three levels of densities are set: dense, medium, sparse. The density can be calculated by \( T_i / (T - L) \), where \( T_i \) is the total time of all heats, which can be calculated by \( \sum_{i=1}^{n} (T_{i0} - T_{i1}) \), and \( T \) is the total time of the schedule, which can be calculated by \( C_n - s_n \). \( L \) is the number of used ladle. We define the average density of three practical data sets as medium. It is \((0.208+0.206+0.258)/3=0.224\). When the density is greater (smaller) than 0.5 (0.2) of medium one, we define it as dense (sparse). We also assume that each interval is normally distributed, but in order to create these three kinds of data sets, we have to adjust its upper bounds, lower bounds, average and standard deviation. In this case, they are \((4000, 1000, 3000, 400), (6000, 1000, 5800, 100) \) and \((4000, 8000, 7000, 100) \) for dense, medium and sparse. Now we can follow the strategy and insert all valid heats and intervals, which confirmed constrains, into limited ladles.

4.2.2. The Experimental Results and Analysis

After 100 simulation data sets for each density and the number of heats had been generated, we applied our algorithm on these data sets and obtained the statistic result (see Table 2). “The number of ladles” is the number of ladles we used in simulation data set. “Density” is the average density of 100 data sets. “CPU(s)” is the average time the algorithm spends. “Ladle(%)” is the ladle successful optimization rate. In Section 3.3, we can know we have to use no less than the ladles used in the schedule created via the algorithm, thus we record the percentage of the data set that the ladles used in the schedule created via the algorithm, which is reflected by the percentage of the number of optimization. “Interval(%)” is the average interval optimization rate, which can be calculated by \( \sum_{i=1}^{n} (T_{i0} - T_{i1}) / (T_{i0} - T_{i1}) \) when \( n \) is the number of data sets, \( T_{i0} \) is the total interval time of simulation data set \( i \), \( T_{i0} \) is the total interval time of simulation data set \( i \) optimized by algorithm in Section 3.2, \( T_{i0} \) is the total interval time of simulation data set \( i \) optimized by 3-Step. If \( T_{i0} - T_{i1} = 0 \), the schedule cannot be optimized, we define \( (T_{i0} - T_{i2}) / (T_{i0} - T_{i1}) \) as 1. We consider the intervals of data set optimized by algorithm in Section 3.2 as a lower bound because it neglects the extra constrains and the effect of the 3-Step algorithm can not better than that. The ratio of the optimized interval time and interval time might be optimized can reflect the performance better because the percentage of interval time that can not be optimized is quite different from each data set, which may greatly affect the optimized rate. The last two values are based on estimation, the actual effect may be better.

In the results, we can easily find that the optimization effect is significant. Both ladles and intervals are greatly optimized at all density. If the number of heats is no more than 100 (a ladle schedule for one day), the length of response time can be accepted by user.

5. Conclusion

This study presents a fast heuristic algorithm based on VRPTW model for the ladle scheduling problem, which is a very important problem in the steelmaking and continuous casting process. The problem and the model are quite different from various previous vehicle routing problems because it needs to minimize the ladle number and the interval time of adjacent heats in each path. That is also the main target of the problem. According to the notable characteristic of the problem and based on the thought of branch and bound, we use different strategy in each step to optimize the problem gradually instead of searching in breadth-first search or other high time complexity search algorithm. Comparing to previous algorithm, it has three superiorities. First is optimization result. Second is computational speed. Third is stability. Therefore, the algorithm is more reliable and stronger than the previous ones. In simulation experiment, the results show its good performance under any circumstance. It may help the steel plants to improve the production efficiency, reduce energy consumption and build a steelmaking automatic system. In addition, the thought can be used to solve similar problem in other fields.

Acknowledgements

This article has been finished in Northeastern University. We would like to express our gratitude to our supervisor Prof. Shi-Xin Liu for his help and suggestion and Dr. Yuan-Yuan Tan for providing all materials in her study in NEU. They have greatly inspired us. Besides, the article is revised in Tsinghua University. I would like to thank my supervisor Prof. Yan-da Li and Dr. Xiao-wo Wang for their valuable comments and experiment environment support in THU. The research is financially supported by National Natural Science Foundation of China (71171038, 61333006).

Table 2. The performance of 3-Step algorithm applied on different density of simulation data.

<table>
<thead>
<tr>
<th>number of heats</th>
<th>number of ladles</th>
<th>density</th>
<th>item</th>
<th>dense</th>
<th>medium</th>
<th>sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>10</td>
<td>density</td>
<td>0.601</td>
<td>0.384</td>
<td>0.142</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU(s)</td>
<td>0.031</td>
<td>0.029</td>
<td>0.024</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ladle(%)</td>
<td>&gt;99</td>
<td>&gt;98</td>
<td>&gt;99</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>interval(%)</td>
<td>&gt;79</td>
<td>&gt;75</td>
<td>&gt;84</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>density</td>
<td>0.592</td>
<td>0.362</td>
<td>0.114</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU(s)</td>
<td>0.321</td>
<td>0.22</td>
<td>0.192</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ladle(%)</td>
<td>&gt;99</td>
<td>&gt;99</td>
<td>&gt;99</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>interval(%)</td>
<td>&gt;72</td>
<td>&gt;85</td>
<td>&gt;88</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>15</td>
<td>density</td>
<td>0.618</td>
<td>0.375</td>
<td>0.139</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU(s)</td>
<td>15.173</td>
<td>18.941</td>
<td>15.668</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ladle(%)</td>
<td>&gt;99</td>
<td>&gt;97</td>
<td>&gt;94</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>interval(%)</td>
<td>&gt;92</td>
<td>&gt;64</td>
<td>&gt;68</td>
<td></td>
</tr>
</tbody>
</table>

REFERENCES