On-Line Scheduling with Non-Crossing Constraints

Lele Zhang, Kwanniti Khammuang, Andrew Wirth *

Department of Mechanical Engineering, The University of Melbourne, VIC 3010, Australia

Abstract

We consider the problem of on-line scheduling with non-crossing constraints. The objective is to minimize the latest completion time. We provide optimal competitive ratio heuristics for the online-list and online-time problems with unit processing times, and a 3-competitive heuristic for the general online-time problem.

Key words: scheduling; on-line; crane; non-crossing constraint

1 Introduction

We consider the on-line version of the scheduling with non-crossing constraints problem. Such constraints occur, for example, when quay cranes are scheduled to load and unload containers at port terminals, as crane arms cannot cross each other. The resulting crane and vessels assignments play an essential role in port terminal management. Non-crossing constraints also appear in manufacturing when material is moved by cranes operating on a common track.

The crane scheduling problem for container terminals was first investigated by Daganzo [1]. He specified a common crane working environment: ships are divided into holds (jobs) and at most one crane can work on a hold at a time. The objective is to minimize the aggregate cost of ship delay with berth length limitations. The paper presented an algorithm to determine the number of cranes handling the bays. Nevertheless, he did not consider the interference between quay cranes.

* Corresponding author. Tel.: +61-3-8344 4852; Fax: +61-3-9347 8784 Email address: wirtha@unimelb.edu.au (Andrew Wirth).
Lim et al. [2–6] devoted much of their attention to the problem of off-line crane scheduling with various constraints. In their earlier papers [2,3], they attacked the problem of maximizing the total throughput under several spatial constraints. Lim et al. [4] established a fully polynomial time approximation scheme for the NP-hard problem of minimizing the latest completion time under the non-crossing constraint, with a fixed number of cranes (greater than one). They also constructed three 2-competitive algorithms, for the problem with an arbitrary number of cranes.

The problem of crane scheduling has also been studied in a manufacturing context, where cranes travel on a common track and so cannot move past each other. See, for example, Lieberman and Turksen [7], Lieberman and Turksen [8], and Hooker et al. [9]. In particular they address problems with release times.

We shall, in the main, follow the formulation of Zhu and Lim [6], in our analysis of the on-line version of the problem. We assume that there are \( m \) processors and \( n \) jobs. Furthermore, at most one processor may process a job at any one time. Processors are constrained to move in a line and are labeled 1, 2, 3, ... \( m \) according to their relative positions, thus processor \( k \) is to the left of processor \( l \) if \( k < l \). Furthermore, the processors’ relative positions must remain unchanged throughout the scheduling process. Job \( i \) is characterized by its processing time \( p_i \) and location \( x_i \). Without loss of generality we assume that \( 0 < x_i < 1 \) and also assume that \( x_i \neq x_j \) if \( i \neq j \). Let \( s_i \) denote the time when processing of job \( i \) starts. The objective is to minimize the makespan, the latest completion time of all the jobs, \( C_{\text{max}} = \max_i(s_i + p_i) \). The non-crossing constraint states that, if at some point in time two processors are processing two jobs, then the processor which is on the left of the other must process the job that is on the left of the other job, that is the job with the smaller location value. In other words, the constraint may be stated as: if job \( i \) is assigned to processor \( k \) and job \( j \) is assigned to processor \( l \) and their processing time intervals overlap then \( x_i < x_j \) if and only if \( k < l \). (See Figure 1, where \( J_i \) and \( P_j \) denote the \( i \)th job and the \( j \)th processor, respectively.) In the standard scheduling notation [10], our problem could be denoted as \( P_m|\text{non-crossing}|C_{\text{max}} \).

**Insert Figure 1**

Traditionally scheduling models have assumed that all problem information is available prior to any decision making. More recently on-line versions of various scheduling problems have been studied. Indeed, a major review of container terminal operations, Steenken et al. [11] (page 16) proposes that such problems demand an on-line approach. In the manufacturing context we may identify a processor with a crane, and in the port terminal one we may, if necessary, identify a processor with the set of cranes working on one ship.
However, to the best of our knowledge, there are no earlier on-line accounts of the non-crossing constraint problem.

A discussion of various on-line paradigms may be found, for example, in Pruhs et al. [10]. The two most common paradigms are one-by-one (on-line list) and release dates (on-line time), discussed in more detail below. Two evaluation measures for an on-line algorithm $A$ are its competitive performance ratio $R_A = \inf \{ r \geq 1 : A(I)/OPT(I) \leq r, \text{ for all } I \}$ where $OPT$ denotes the optimal off-line solution, and $I$ is a problem instance and asymptotic performance ratio $R_A^\infty = \inf \{ r \geq 1 : \text{ for some } N > 0, A(I)/OPT(I) \leq r, \text{ for all } I \text{ with } OPT(I) \geq N \}$. The latter measure may avoid certain small case anomalies.

2 One-By-One ($P_m | \text{non-crossing} | C_{\text{max}}$)

We recall the definition of one-by-one (or on-line list) scheduling as it applies to our problem.

We are presented with a list of $n$ jobs $(p_1, x_1), \ldots, (p_n, x_n)$ and we must allocate processors and starting times for these jobs, one by one without any knowledge of the remaining jobs in the list. Let $c_A(i)$ denote the processor assigned to job $i$ by algorithm $A$. Our objective is to minimize the makespan, $C_{\text{max}}$, the latest completion time of all the jobs. We shall say that an algorithm does not allow any unnecessary idle time if it allocates each job to the earliest possible starting time. We shall assume that none of our one-by-one algorithms allow unnecessary idle time.

We introduce the following

**Middle Heuristic ($M$)**

Assign the middle of the first available processors. In case of a tie assign the right processor (that is, the processor with a greater index).

**Example** Suppose that $m = 6$ and $n = 8$ with jobs: $(1.0, 0.7), (4.0, 0.5), (1.5, 0.6), (1.9, 0.3), (2.8, 0.1), (0.8, 0.9), (1.0, 0.2), \text{ and } (2.0, 0.05)$. The middle heuristic $M$ then produces the schedule shown in Figure 2.

Insert Figure 2

**Proposition 1** $M$ is optimal for the unit processing time problem, with $R_M^\infty(p_i = 1) = \frac{m}{\lfloor \log_2(m+1) \rfloor}$. For the general case with arbitrary processing times $R_A^\infty = m$ for any algorithm $A$.

**Proof.**
Case (i): Unit processing times

We begin by showing that $R^\infty_A(p_i = 1) \geq \frac{m}{\log_2(m+1)}$ for any $A$. Choose any $x_1$ in $(0,1)$. By the above comment about idle time we may assume that $A$ assigns job 1 ($p_1, x_1$) to some processor $c_A(1)$ in time interval $[0,1]$. We now iteratively construct the following sequence of partitions, $A(i,0)$, of the job location interval $(0,1)$. Firstly let $A(1,0) = \{x_1\}$. We say that the, as yet, unassigned processors 1, 2, ..., $c_A(1) - 1$ are contained in the interval $(0, x_1)$ and the unassigned processors $c_A(1)+1, ..., m$ are contained in the interval $(x_1, 1)$. Consider the two subintervals $(0, x_1)$ and $(x_1, 1)$. Let $x_2$ belong to the subinterval containing the fewest unassigned processors, with a tie being broken arbitrarily. (If one of these two subintervals contains no unassigned processors we replace the interval $(0, 1)$ by that subinterval, as discussed below.) We then repeat this process in the following way. We let $A(i,0) = \{x_j : \text{job } j \text{ is assigned to be processed by } c_A(j) \text{ in time interval } [0,1], 1 \leq j \leq i\}$. Thus, $A(2,0) = \{x_1, x_2\}$. We may consider the elements of $A(i,0)$ as end-points of a partition of $(0,1)$ into subintervals. Let $x_{i+1}$ belong to a subinterval containing the fewest unassigned processors. Repeat this process until we have a subinterval $(a_1, b_1)$ with no unassigned processors. Suppose the last job assigned so far is job $k$. Now replace the position interval $(0, 1)$ above, by $(a_1, b_1)$ and choose the next location $x_{k+1}$ to lie in $(a_1, b_1)$. $A$ will assign job $k + 1$ to the time interval $[1, 2]$ and we let $A(i,1) = \{x_j : \text{job } j \text{ is assigned to be processed by } c_A(j) \text{ in time interval } [1,2], 1 \leq j \leq i\}$. Repeat for $[a_2, b_2]$ and so on.

It is easy to show that if $m < 2^q - 1$ for some integer $q$ then at best $q - 1$ processors are working at any one time. Thus, $R^\infty_A(p_i = 1) \geq \frac{m}{\log_2(m+1)}$.

On the other hand, for the middle heuristic with $2^q - 1$ processors, we have $q$ processors working at any one time. Hence $R^\infty_M(p_i = 1) = \frac{m}{\log_2(m+1)}$. That is, for the problem with unit processing times $M$ is an optimal algorithm.

Case (ii): Arbitrary processing times

We now show that for the case of arbitrary processing times all algorithms are asymptotically $m$-competitive, that is $R^\infty_A = m$.

We construct the following sequence of jobs (for $m \geq 2$).

Step 0: Let $s = 0$, $x_0 = 1$.

Step 1: Let $\epsilon$ be a positive number such that $\epsilon \to 0$. Then let $p_{s+i} = \epsilon$ for $i = 1, \ldots, k$, $x_s > x_{s+1} > x_{s+2} > \ldots > x_{s+k}$ with the sequence continued until the first of processors 1 or 2 is chosen. That is, $c_A(s+k) = 1$ or 2.

If $c_A(s+k) = 2$ then Step 2 is executed else Step 3 is executed.

Step 2: Let $x_{s+k+1} < x_{s+k}$ and $p_{s+k+1} = 1$, hence $c_A(s+k+1) = 1$. Replace $s$ by $s+k+1$ and return to Step 1.
Step 3: Let \( p_{s+k+i} = 2\epsilon \) for \( i = 1, \ldots, k' \), \( x_{s+k-1} > x_{s+k+1} > x_{s+k+2} > \ldots > x_{s+k+k'} > x_{s+k} \) with the sequence continued until processor 2 is chosen. That is, \( c_A(s + k + k') = 2 \). Let \( p_{s+k+k'+1} = 1 \), and \( x_{s+k+k'+1} < x_{s+k} \) hence \( c_A(s + k + k' + 1) = 1 \). Replace \( s \) by \( s + k + k' + 1 \) and return to Step 1.

We repeat this process until \( m \) jobs of size 1 have been generated. (Note that \( k \) and \( k' \) depend on the particular algorithm iteration of a step.) So, \( R^\infty_A \geq m \). If a heuristic \( A \) does not allow unnecessary idle time then \( C_{max}(I) \leq \sum_{i=1}^{n} p_i \). Clearly, \( OPT(I) \geq \sum_{i=1}^{n} p_i \), so \( R^\infty_A = m \). □

3 Release Dates \((P_m|\text{non-crossing}, r_i |C_{max})\)

In this section, we consider the scheduling problem of jobs arriving over time. That is, we now assume that each job, \((p_i, x_i, r_i)\), is described by a release date \( r_i \) as well as a processing time and a position. Once again, each job must be scheduled to a processor and time slot. However, the scheduling decision may be delayed, that is, when a job is released, the scheduler may assign it immediately or wait, for example, until a later time or the arrival of the next job (if any).

3.1 Unit Processing Times \((p_i = 1)\)

First we consider the equal length job problem \( p_i = 1 \) for \( m \geq 2 \).

**Proposition 2** For the unit processing time problem with \( m \geq 2 \) no on-line algorithm has a competitive ratio better than \( \frac{\sqrt{5}+1}{2} \).

**Proof.** Suppose that an on-line algorithm decides to schedule the first job \((1, x_1, 0)\) at time \( S \) to processor \( c_A(1) \). Now suppose that either no further jobs arrive or \( m - 1 \) jobs, whose positions are greater than \( x_1 \), if \( c_A(1) > \left\lfloor \frac{m}{2} \right\rfloor \) (or smaller than \( x_1 \), if \( c_A(1) \leq \left\lfloor \frac{m}{2} \right\rfloor \)), arrive at time \( S+\epsilon \), where \( \epsilon > 0 \). In the former case \( C_{max} = S + 1 \), while in the latter, \( C_{max} \geq S + 2 \). The optimal makespans are 1 and \( S + 1 + \epsilon \), respectively. So \( R_A \geq \lim_{\epsilon \to 0} \inf \max \{ \frac{S+1}{1}, \frac{S+2}{S+1+\epsilon} \} \). By the elementary algebra, it follows that the minimum of \( R_A \) is achieved when \( S = S_0 = \frac{\sqrt{5}-1}{2} \), and thus \( R_A \geq \frac{\sqrt{5}+1}{2} \). □

Next we introduce a new heuristic for this special case and further discuss its competitive performance.

**Equal Heuristic (E)**
Whenever a job is either released or completed carry out the following procedure. Do not schedule until all processors are idle. If the current time $t$ is smaller than $S_0 = \sqrt{5} - 1$, wait until $t = S_0$. Then assign the leftmost available processor to an unscheduled job with the smallest position. Continue the assignment until either no processors or no jobs remain available.

**Lemma 3** Without loss of generality, we may assume that, for any $E$ schedule, at least one processor is not idle throughout time interval $[S_0, C_{\text{max}}^E(I)]$.

**Proof.** The proof is similar to that in Observation 1 in [12]. We show that if there is an idle subinterval in $[S_0, C_{\text{max}}^E(I)]$ for some instance $I$, where $C_{\text{max}}^E(I)$ is the makespan of the $E$ schedule for $I$, then there is another $I'$ for which $R_E(I') > R_E(I)$ and such that $[S_0, C_{\text{max}}^E(I')]$ contains no idle subinterval. Suppose that there is an idle subinterval $(d, f)$ ($S_0 \leq d < f < C_{\text{max}}^E(I)$), in which none of the $m$ processors work.

**Case (i):** $d = S_0$

In this case, the release time of the first job $r_1$ must be equal to $f$. We may reduce the release time of each job in instance $I$ by $f - S_0$ to form another instance $I'$. It is clear that the makespan of the $E$ schedule and the optimal solution for the new instance will both decrease by $f - S_0$. Furthermore, the competitive ratio of the new instance $I'$ is greater than $\frac{C_{\text{max}}^E(I')}{OPT(I)}$. So we may assume that there is no idle interval at the beginning of time interval $[S_0, C_{\text{max}}^E(I)]$.

**Case (ii):** $d > S_0$

According to the scheduling rules of heuristic $E$, jobs processed before $d$ do not affect the processing of the successive jobs, while the successive jobs processed after $f$ must arrive at or after $f$. We may delete all jobs which are completed before or at $d$, without affecting the makespans of the $E$ schedule or the optimal solution. Then we reduce the release times of the remaining jobs by $f - S_0$, respectively. As the makespans both decrease by $f - S_0$, the new instance $I'$ has a competitive ratio greater than $\frac{C_{\text{max}}^E(I')}{OPT(I)}$.

We can conclude that, without loss of generality, we may assume that there exists no idle subinterval in time interval $[S_0, C_{\text{max}}^E(I)]$. □

**Proposition 4** Heuristic $E$ is optimal for the unit processing time problem with $m \geq 2$.

**Proof.** Consider an arbitrary instance $I$. In the $E$ schedule of $I$, processors once assigned to some jobs must start and finish the jobs simultaneously, because $p_i = 1 \forall i$. Given that we consider the worst case performance of $E$, by Lemma 3, we assume that there is no idle period in $[S_0, C_{\text{max}}^E(I)]$, and
thus there is at least one processor working at any time of the interval. Since processor 1, the leftmost processor, is always the first assigned processor, it is working all the time during \([S_0, C_{\text{max}}^E(I)]\). Then we divide the makespan interval \([0, C_{\text{max}}^E(I)]\) into \(c+1\) time subintervals \(I_k (k = 0, 1, \ldots , c)\), where \(c\) is an integer. In each subinterval \(I_k\), except the first one which is purposely kept idle, processor 1 works on a single job. So \(I_k = [0, S_0)\) for \(k = 0\), and \(S_0 + k - 1, S_0 + k\) for \(1 \leq k \leq c\). Let \(B_k\) be the number of working processors in \(I_k\). Obviously, \(B_0 = 0, 1 \leq B_k \leq m\) for \(1 \leq k \leq c\) and \(C_{\text{max}}^E(I) = S_0 + c\). Let \(I_l (l \neq c)\) be the last interval during which not all processors are working, that is, \(l = \max\{k, B_k < m\) and \(0 \leq k \leq c - 1\}\).

If \(l = 0\), then \(m\) processors are working all the time during time interval \([S_0, S_0 + c - 1]\). Since \(p_i = 1 \forall i\), we have the number of jobs \(n = (c-1)m + B_c\). Since \(1 \leq B_c \leq m\), \(\text{OPT}(I) \geq \left\lceil \frac{n}{m} \right\rceil = \left\lceil \frac{(c-1)m + B_c}{m} \right\rceil = c\). So \(\frac{C_{\text{max}}^E(I)}{\text{OPT}(I)} \leq \frac{S_{n+c}}{c} \leq 1 + S_0 = \frac{\sqrt{5} + 1}{2}\). Otherwise, if \(l \geq 1\), then there are \((c-l-1)m + B_c\) jobs processed after \(S_0 + 1\) and they must arrive after \(S_0 + l - 1\). Consider a lower bound of an optimal solution. \(\text{OPT}(I) \geq S_0 + l - 1 + \left\lceil \frac{(c-l-1)m + B_c}{m} \right\rceil = S_0 + c - 1\). Since \(c \geq 1\) and \(c\) is an integer, \(c \geq 2\) and \(\frac{C_{\text{max}}^E(I)}{\text{OPT}(I)} \leq \frac{S_{n+c}}{S_0 + c - 1} \leq 1 + \frac{1}{S_0 + 1} = \frac{\sqrt{5} + 1}{2}\). By Proposition 2, \(R_A \geq \frac{\sqrt{5} + 1}{2}\). Hence the result follows. \(\Box\)

### 3.2 Arbitrary Processing Times

Now we consider the corresponding problem with arbitrary processing times.

**Proposition 5** No on-line algorithm has a competitive ratio better than 2 for the problem with arbitrary processing times and \(m \geq 2\).

**Proof.** Consider the following job sequence. \(m - 1\) jobs arrive at time 0 with unit processing time, that is, \(p_i = 1\) and \(r_i = 0\) for \(1 \leq i \leq m - 1\). Suppose that an on-line algorithm \(A\) schedules job \(i\) at time \(S_i\).

1. If \(\max S_i \geq 1\), then \(C_{\text{max}} \geq 2\) and \(\text{OPT} = 1\). So, obviously, \(R_A \geq 2\).
2. If the \(m - 1\) jobs are not all placed on separate machines, then there is at least one processor assigned to more than one job. Hence \(C_{\text{max}} \geq 2\) and \(\text{OPT} = 1\). We have \(R_A \geq 2\).
3. Suppose that \(\max S_i < 1\). If all the \(m - 1\) jobs are processed on different machines, then at least one of the processors 1 and \(m\) is not idle. Without loss of generality, we assume that processor 1 is not idle and job \(g\) is processed on it and started at \(S_g\). At time \(S_g + \epsilon\), where \(\epsilon > 0\) and \(\epsilon \to 0\), the \(m\)th job is released with \(p_m = 1 - S_g\) and \(x_m < x_g\). At best \(A\) can complete job \(m\) at \(S_g + 1 + 1 - S_g = 2\). An optimal solution finishes all \(m\) jobs at \(S_g + \epsilon + 1 - S_g = 1 + \epsilon \to 1\), as \(\epsilon \to 0\). It is apparent that \(R_A \geq 2\).
The result follows. □

For the general problem, we generate a new heuristic and then analyze its performance.

**Large-Job-First Heuristic (L)**

Do not schedule until all processors are idle. Rearrange the current available jobs in non-increasing processing time order and let $U_t$ denote the job set in which $p_1^t \geq p_2^t \ldots \geq p_{N_t}^t$, where $t$ is the current time and $N_t$ is the number of current available jobs. Assign the leftmost available processor to $J_i^*$ with the smallest position, where $J_i^* \in U_t$ and $1 \leq i \leq \min\{m, N_t\}$. Continue the assignment until no processors or no jobs are available.

**Lemma 6** Without loss of generality, we may assume that, for any $L$ schedule, at least one processor is not idle throughout the makespan.

**Proof.** An argument analogous to that in Lemma 3 yields the result. □

For $m = 2$, clearly by Lemma 6, $R_L(m = 2) \leq 2$ and by Proposition 5 $R_L(m = 2) \geq 2$. So $R_L(m = 2) = 2$.

**Proposition 7** $3 - \frac{2}{m} \leq R_L < 3$ for $1|\text{non-crossing}, r_i|C_{\text{max}}$ with $m \geq 3$.

**Proof.** First we show that $R_L \geq 3 - \frac{2}{m}$. Consider the following job sequence $\Pi$: $(G, 0.3, 0), (G, 0.1, \epsilon), (1, 0.3 + \epsilon, \epsilon), (1, 0.3 + 2\epsilon, \epsilon), \ldots (1, 0.3 + x\epsilon, \epsilon)$, where $G$ is a large finite integer, $\epsilon$ is a small positive number ($\epsilon \to 0$) and $x = \frac{(m-2)}{m}$. Insert Figure 3

For the schedule of heuristic $L$, as Figure 3 shows, the makespan satisfies

$$C_{\text{max}}^L(\Pi) = 2G + \left[\frac{x - (m-1)}{m}\right] = 2G + \left[\frac{(m-2)G - (m-1)}{m}\right] \geq 3G - \frac{2}{m}G - 1.$$ 

As for the optimal solution, processor 1 is assigned to job 2 and processor 2 is assigned to job 1. Also, processors 3, 4, \ldots, $m - 1$ and $m$ are assigned to the remaining jobs in an obvious way. Thus, $OPT(\Pi) = G + \epsilon$. Then, we have

$$\frac{C_{\text{max}}^L(\Pi)}{OPT(\Pi)} \geq \frac{3G - \frac{2}{m}G - 1}{G + \epsilon} \to 3 - \frac{2}{m},$$ as $G$ is large enough and $\epsilon \to 0$.

Now we show that $R_L < 3$. 

8
Given an instance $I$, we divide the makespan interval $(0, C_{\text{max}}^L(I))$ of its $L$-schedule into $c$ time subintervals $I_k = (a_{k-1}, a_k]$ for $k = 1, 2, \ldots, c$, where $c$ is an integer and $a_{k-1}$ is the time when processor 1 starts to process some new job. Suppose that the job with the latest release date is released during $I_t = (a_{t-1}, a_t]$ for some $t \leq c - 1$. Consider the following subsets:

- $U_a$: the jobs processed in $I_t$;
- $U_b$: the jobs released before or at $a_t$ and processed after $a_t$;
- $U_{a_t-1}$: the jobs released after $a_{t-1}$ and processed after $a_t$;
- $U_{a_t}$: the jobs released before or at $a_t$ and processed after $a_{t-1}$, $U_{a_t} = U_a \cup U_b$;
- $U_{a_t}$: the jobs processed after $a_t$, $U_{a_t} = U_a \cup U_b$.

Let $N_a, N_b, N_{a_t-1}$ and $N_{a_t}$ denote the numbers of jobs and $p_i^b, p_i^l, p_i^b, p_i^l$ and $p_i^l$ denote the processing times of jobs $h, i, j, k$ and $l$ in the subsets $U_a, U_b, U_{a_t-1}$ and $U_{a_t}$, respectively. We arrange the jobs of each subset in non-increasing processing time order.

In preparation for the establishment of the upper bound of $L$’s competitive ratio, we first provide three lower bounds for the optimal makespan.

\[ \text{OPT}(I) \geq \min \{ \sum_{i=1}^{N_h} p_i^b \} > a_{t-1} + \sum_{i=1}^{N_h} p_i^b. \]  

\[ \text{OPT}(I) \geq \max \{ \sum_{i=1}^{N_h} p_i^b \} \geq \min \{ \sum_{i=1}^{N_h} p_i^b \} > a_{t-1} + \sum_{i=1}^{N_h} p_i^b. \]  

\[ \text{OPT}(I) \geq a_t - a_{t-1}. \]  

The last lower bound (3) above is derived from Lemma 6, since there is no idle interval during the makespan and thus at least one processor is working on a single job all the time during $I_t$.

Considering the $L$ schedule of instance $I$, we have two cases below.

**Case (i): $U_a = \emptyset$**

As $U_a = \emptyset$, we have $U_{a_t} = U_b$ and $p_i^b = p_i^l$ for $1 \leq i \leq N_b$. Hence, the maximal completion time is given by

\[ C_{\text{max}}^L(I) = a_t + p_i^b + p_{i+1}^b + \ldots + p_{v_m + 4}^b, \]  

where $v = \lceil \frac{N_b}{m} \rceil - 1$.  

If $v \geq 1$, then, since $p_{jm+i}^b \geq p_{(j+1)m+1}^b$ where $0 \leq j \leq v - 1$ and $1 \leq i \leq m$, the following inequality must be satisfied:

\[ \sum_{i=1}^{N_h} p_i^b \geq \frac{mp_{m+1}^b + mp_{2m+1}^b + \ldots + mp_{v_m+1}^b + p_{v_m+1}^b + \ldots + p_{N_b}^b}{m} > p_{m+1}^b + p_{2m+1}^b + \ldots + p_{v_m+1}^b. \]
Otherwise, if \( v = 0 \), then the right-hand side of (5) is zero whilst the left-hand side is a positive value. Therefore, (5) is also satisfied for the case of \( v = 0 \).

By inequalities (1), (2), (3), (4) and (5), we have

\[
\frac{C^L_{\text{max}}(I)}{\text{OPT}(I)} \leq \frac{a_t + p^b_1 + p^b_{m+1} + \ldots + p^b_{vm+1}}{\frac{1}{3}(a_{t-1} + p^b_1 + a_t - a_{t-1} + a_t - a_{t-1} + \sum_{i=1}^{vm+1} p^b_i)} < \frac{3(a_t + p^b_1 + p^b_{m+1} + \ldots + p^b_{vm+1})}{a_t + a_{t-1} + p^b_1 + p^b_{m+1} + p^b_{2m+1} + \ldots + p^b_{vm+1}} \leq 3.
\]

**Case (ii):** \( U_a \neq \emptyset \)

For convenience, we introduce three variables and define them as follows: \( \alpha = p^a_1 + p^a_{m+1} + \ldots + p^a_{am+1} \), \( \beta = p^b_1 + p^b_{m+1} + \ldots + p^b_{vm+1} \) and \( \gamma = p^a_1 + p^a_{m+1} + \ldots + p^a_{wm+1} \), where \( u = \lceil \frac{Na}{m} \rceil - 1 \), \( v = \lceil \frac{Na}{m} \rceil - 1 \) and \( w = \lceil \frac{Na}{m} \rceil - 1 \). Then the makespan is given by

\[
C^L_{\text{max}}(I) = a_t + p^a_1 + p^a_{m+1} + \ldots + p^a_{wm+1} = a_t + \gamma.
\] (6)

From the scheduling mechanism of heuristic \( L \), we have \( p^a_i = p^+_i \) for \( 1 \leq i \leq m \) and \( p^a_j = p^+_{m+j} \) for \( 1 \leq j \leq Na \). Since \( p^+_{jm+1} \geq p^+_{(j+1)m+1} \), where \( 1 \leq j \leq u - 1 \) and \( 1 \leq i \leq m \), we have

\[
\sum_{i=1}^{Na-1} p^+_i \geq \frac{mp^+_m + mp^+_2m + \ldots + mp^+_{(a-1)m+1} + p^+_{(a+1)m+1} + \ldots + p^+_Na}{m} = p^a_1 + p^a_{m+1} + \ldots + p^a_{um+1} = \alpha.
\] (7)

By inequalities (5) and (7), we obtain

\[
\frac{\text{OPT}(I)}{a_t + \gamma} > \frac{\sum_{i=1}^{Na-1} p^+_i + \sum_{j=1}^{N_b} p^b_j}{m} \geq \alpha + \beta - p^b_1.
\] (8)

Given that \( U_{a_i} = U_a \cup U_b \), we have \( \gamma \leq \alpha + \beta \). (Refer to Lemma 8 for the proof in detail.) Also, by (2), (3), (6) and (8), the result below bounds the competitive ratio of heuristic \( L \).

\[
\frac{C^L_{\text{max}}(I)}{\text{OPT}(I)} < \frac{a_t + \gamma}{\frac{1}{3}(a_{t-1} + p^b_1 + a_t - a_{t-1} + \alpha + \beta - p^b_1)} = \frac{3(a_t + \gamma)}{a_t + \alpha + \beta} \leq 3.
\]

The proof is now complete. \( \Box \)
Lemma 8 $\gamma \leq \alpha + \beta$.

**Proof.** First we rearrange sets $U_a$ and $U_b$ such that the values of $\alpha$, $\beta$ and $\gamma$ remain the same. The rearrangement is stated as follows. We let $K = \max\left(\frac{N_a}{m}, \frac{N_b}{m}\right)$ and add $Km - N_a$ and $Km - N_b$ dummy jobs with zero processing times into $U_a$ and $U_b$, respectively. That is, both of the sets have $Km$ jobs.

Now we show that $\gamma \leq \alpha + \beta$ by induction.

Since $p_1' = \max(p_1^a, p_1^b)$, without loss of generality, we assume that $p_1^a \geq p_1^b$. Then, $p_1^a = p_1^b$. For $K = 1$, at least one of the jobs $2, 3, \ldots, m + 1$ in $U_a$, comes from $U_b$. It follows that $p_{m+1}^* \leq p_1^b$. So $p_1^a + p_{m+1}^* \leq p_1^a + p_{m+1}^b$. Thus, $\gamma \leq \alpha + \beta$ is valid for $K = 1$. Assume that $p_1^a + p_{m+1}^* + \ldots + p_{(2k-1)m+1}^* \leq p_1^a + \ldots + p_{(k-1)m+1}^b + p_{(k-1)m+1}^b$ for $K = k \geq 1$. Then we consider the case of $K = k + 1$. Let a superscript $'$ indicate the case of $K = k + 1$. Then the compositions of the sets are presented below.

\begin{align*}
U_a: & \quad p_1^a, p_2^a, \ldots, p_{(2k-1)m+m}^a; \\
U_a': & \quad p_1'^a, p_2', \ldots, p_{(2k-1)m+m}^a, p_{2km+1}', \ldots, p_{(2k+1)m+m}'; \\
U_a: & \quad p_1'^a, p_2', \ldots, p_{(k-1)m+m}^a, p_{2km+1}', \ldots, p_{(2k+1)m+m}'; \\
U_a': & \quad p_1', p_2', \ldots, p_{(k-1)m+m}', p_{km+1}', \ldots, p_{km+m}'; \\
Q_a: & \quad U_a' - U_a = p_{km+1}^a, \ldots, p_{km+m}^a; \\
Q_b: & \quad U_b' - U_b = p_{km+1}^b, \ldots, p_{km+m}^b.
\end{align*}

As $U_a = U_a \cup U_b$, it is clear that the difference between the sets $U_a$ and $U_a'$ is $Q_a \cup Q_b$. Therefore, we shall prove that the inequality $\gamma \leq \alpha + \beta$ holds for $K = k + 1$, by showing how the items in the sets $Q_a$ and $Q_b$ are placed in $U_a$ to form $U_a'$. For convenience, we consider each item in the sets as a number.

Since $p_{2km}^* = \min(p_{km}^a, p_{km}^b)$, we assume, without loss of generality, that $p_{km}^a \leq p_{km}^b$. Thus, all numbers in $Q_a$ are placed after $p_{2km}$. We now consider the following cases:

**Case (i):** $p_{km+1}^b \leq p_{km+1}^b$

In this case, all numbers in $Q_a$ and $Q_b$ are put after $p_{2km}^a$. The analogous argument to that when $K = 1$ leads to that $p_{2km+1}^a + p_{(2k+1)m+1}^a \leq p_{km+1}^a + p_{km+1}^b$. Consequently, $\gamma \leq \alpha + \beta$ for $K = k + 1$.

**Case (ii):** $p_{km+1}^b > p_{km+1}^b$

Suppose that $p_{km+1}^b$ is put in position $F$, where $F \leq 2km$. Numbers before $F$ remain unchanged, while numbers at or after that position are shifted backwards by at most $m$. So we obtain, for some integer $z$,
\[ p_{zm+1}' = p_{zm+1}^* \quad \text{where } 0 \leq z \leq \left\lceil \frac{F-1}{m} \right\rceil - 1; \]
\[ p_{zm+1}' \leq p_{km+1}^b \quad \text{where } z = \left\lceil \frac{F-1}{m} \right\rceil; \]
\[ p_{(z+1)m+1}' \leq \begin{cases} p_{zm+1}^*, & \text{where } \left\lceil \frac{F-1}{m} \right\rceil \leq z \leq 2k - 1; \\ p_{km+1}^a, & \text{where } z = 2k. \end{cases} \]

Also, by the inductive hypothesis, we have

\[ p_1' + p_{m+1}' + \ldots + p_{(2k+1)m+1}' \leq p_1^* + \ldots + p_{(2k-1)m+1}^* + p_{km+1}^a + p_{km+1}^b \]
\[ \leq p_1^a + \ldots + p_{km+1}^a + p_1^b + \ldots + p_{km+1}^b. \]

It follows that \( \gamma \leq \alpha + \beta \) for \( K = k + 1 \). Hence the result follows. \( \square \)

References


Fig. 1. Non-crossing constraints
Fig. 2. A schedule produced by the middle heuristic $M$
Fig. 3. The $L$ schedule and an optimal schedule for instance $I_1$