Regular Article

Neighborhood Search for Solving Personal Scheduling Problem in Available Time Windows with Split-Min and Deadline Constraints

Trang Hong Son[1,2], **Tran Van Lang**[3], **Nguyen Huynh-Tuong**[1]

[1] HCMC University of Technology, Vietnam
[2] Hoa Sen University, Vietnam
[3] HCMC University of Foreign Laguages - Information Technology, Vietnam

Correspondence: Trang Hong Son, son.tranghong@hoasen.edu.vn

Communication: received 24 August 2021, revised 20 September 2021, accepted 21 September 2021
Online publication: 22 October 2021, Digital Object Identifier: 10.21553/rev-jec.284
The associate editor coordinating the review of this article and recommending it for publication was Prof. Vo Nguyen Quoc Bao.

Abstract– **The scheduling of individual jobs with certain constraints so that efficiency is a matter of concern. Jobs have deadlines to complete, can be broken down but not too small, and will be scheduled into some available time windows. The goal of the problem is to find a solution so that all jobs are completed as soon as possible. This problem is proved to be a strongly NP-hard problem. The implementation of the proposed MILP model using a CPLEX solver was also conducted to determine the optimal solution for the small-size dataset. For large-size dataset, heuristic algorithms are recommended such as First Come First Served (FCFS), Earliest Deadline (EDL), and neighborhood search including Stochastic Hill Climbing (SHC), Random Restart Hill Climbing (RRHC), Simulated Annealing (SA) to determine a good solution in an acceptable time. Experimental results will present in detail the performance among the groups of exact, heuristic, and neighborhood search methods.**

Keywords– **splitting-job, available time-window, deadline constraint, FCFS rule, EDL rule, neighborhood search, hill climbing algorithm, simulated annealing algorithm.**

1 Introduction

In our daily life, each human being has a lot of personal tasks that need to be done (called jobs). Each job has an execution time (called processing time) and a required time to complete (called deadline) for each job. We also have free time slots to which jobs can be scheduled (called available time window) and time slots that are not available or do not need to be scheduled (called unavailable time window). To simplify problem modeling, unavailable time windows are reduced to milestones (referred to as break times). The scheduling personal problem is the problem of arranging jobs in available time windows so that they are effective according to different criteria. The main constraints in this problem are that the jobs can be broken down but cannot be less than a certain threshold (called $split_{\min}$), and the jobs only can be assigned into available time windows. In addition, the problem also has an additional constraint that the completion time of the job must be before the corresponding deadline of that job. This problem aims to find a solution so that all jobs are completed as soon as possible.

According to Graham [1], this scheduling problem is denoted as follows:

$$1|splittable, split_{\min}, available - windows, deadline|C_{\max}$$

Other notations used in the problem are:

- $J = \{J_1, \ldots, J_n\}$ is the set of n jobs.
- J_i is the i^{th} job.
- p_i is the processing time for job J_i.

Table I
Jobs

Jobs	Processing-time	Deadline
J_1	6	18
J_2	8	20
J_3	4	9
J_4	9	34
J_5	11	42

Table II
Windows

Windows	Available-time
W_1	[0,7]
W_2	[7,15]
W_3	[15,25]
W_4	[25,38]
W_5	[38,+∞)

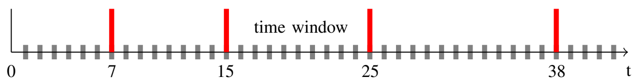


Figure 1. Demonstration of available time windows.

- rj_i is the remaining time for job J_i.
- $\bar{d}_i$ is the deadline for job J_i.
- C_i is the completion time for job J_i.
- $C_{\max}$ is the completion time for all jobs.
- $W = \{W_1, ..., W_m\}$ is the set of m available time-windows.
- W_t is the t^{th} window.
- w_t is the size of window W_t.
- rw_t is the remaining size of the window W_t.
- b_t is the t^{th} break-time.

The problem is illustrated by the input data in the Tables I and II. This simple example has $n = 5$ jobs (J_1, J_2, J_3, J_4, J_5) with the processing time of each job is 6, 8, 4, 9, 11 and the deadline of each job is 18, 20, 9, 34, 42; and $m = 5$ windows (W_1, W_2, W_3, W_4, W_5) with respective available time $[0, 7]$, $[7, 15]$, $[15, 25]$, $[25, 38]$, $[38, +\infty)$; and 4 break times at times $t = 7$, $t = 15$, $t = 25$, $t = 38$ as Figure 1.

Let $split_{\min} = 3$, the possible solutions to the problem are as follows.

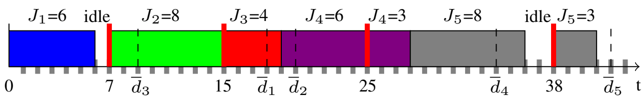


Figure 2. The infeasible solution for violating deadline constraint.

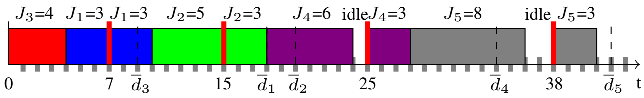


Figure 3. The feasible solution with $C_{\max} = 41$.

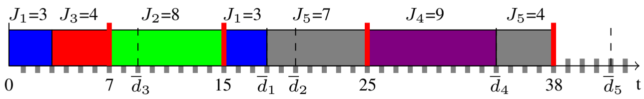


Figure 4. The optimal solution with $C^*_{\max} = 38$.

The optimization personal scheduling problem (PSP) presented at [2] is considered as a special case of this problem with $deadline = \infty$. Because the PSP is a strongly NP-hard problem, this problem under consideration is also a strongly NP-hard problem. In addition, some structural properties of a particular optimal solution are presented at [3], which are:

1) There exists an optimal solution such that there is no idle-time of size $\geq 2 \times split_{\min}$.
2) There exists an optimal solution such that there are only 0 or 1 sub-jobs in a time window.
3) There exists an optimal solution such that there is at most one idle-time in a time window.
4) There exists an optimal solution such that in a time window, if there is idle-time, then idle-time is at the end of the window.
5) There exists an optimal solution such that a job can be split at most S_i, where $S_i = \min\left\{\left\lfloor \frac{p_i}{split_{\min}} \right\rfloor; m\right\}$.
6) There is an optimal solution such that the order of jobs is arranged arbitrarily in a time window.

This problem has the same properties in its optimal solution as the PSP, except for property 6 because this problem is concerned with the order of sub-jobs J_i in the same window W_t.

This paper is organized as follows. The next section shows the mathematical model for this scheduling problem. Section 3 presents the proposed approaches for solving this problem. The results of the experiment are demonstrated in the Section 4. And the final section is the discussion and conclusion of the study.

2 Mathematical Model

Some decision variables are:

- $x_{i,t} \in \{0, 1\}$ is 1 if there exists sub-job J_i assigned to window W_t, otherwise is 0.
- $y_{i,t} \in \mathbb{N}$ is the execution time for the sub-job J_i in the window W_t corresponding to $x_{i,t}$.
- $s_{i,t} \in \mathbb{N}$ is the start time for the sub-job J_i in the window W_t corresponding to $x_{i,t}$.
- $v_{i,j,t} \in \{0, 1\}$ is used to convert from an OR constraint to an AND constraint.

And intermediate variables are:

- $c_{i,t} = s_{i,t} + y_{i,t} \in \mathbb{N}$ is the completion time for sub-job J_i in the window W_t.
- $C_i = \max\limits_{t=1,\ldots,m} (c_{i,t}) \in \mathbb{N}$ is the completion time for job J_i.
- $C_{\max} = \max\limits_{i=1,\ldots,n} (C_i) \in \mathbb{N}$ is the completion time for all jobs.

The Mixed Integer Linear Programming (MILP) model is represented as follows.

Objective function: $\min(C_{\max})$

Subject to:

$$\sum_{t=1}^{m} y_{i,t} = p_i; \forall i = 1, \ldots, n \quad (1)$$

$$\sum_{i=1}^{n} y_{i,t} \leq w_t; \forall t = 1, \ldots, m \quad (2)$$

$$split_{\min} \times x_{i,t} \leq y_{i,t} \leq p_i \times x_{i,t}; \quad \forall i = 1, \ldots, n; \forall t = 1, \ldots, m \quad (3)$$

$$b_t \times x_{i,t} \leq s_{i,t} \leq INF \times x_{i,t}; \quad \forall i = 1, \ldots, n; \forall t = 1, \ldots, m \quad (4)$$

$$b_t \leq s_{i,t} \leq b_{t+1} - y_{i,t}; \quad \forall i = 1, \ldots, n; \forall t = 1, \ldots, m \quad (5)$$

$$\begin{cases} c_{i,t} - s_{j,t} \leq INF \times v_{i,j,t}; & \forall i, j = 1, \ldots, n | i \neq j; \forall t = 1, \ldots, m \\ c_{j,t} - s_{i,t} \leq INF \times (1 - v_{i,j,t}); & \forall i, j = 1, \ldots, n | i \neq j; \forall t = 1, \ldots, m \end{cases} \quad (6)$$

$$C_i \leq \bar{d}_i; \forall i = 1, \ldots, n \quad (7)$$

The constraints are described as below:

- Constraint (1): the total execution time for sub-jobs is equal to the completion time for this job.
- Constraint (2): the total execution time for sub-jobs in a window must not exceed the size of this window.
- Constraint (3): if there is a sub-job assigned in a window, the execution time for this sub-job must be greater than equal to $split_{\min}$ and less than equal to processing time for this job; in addition, this constraint also ensures that if $x_{i,t} = 0$ then $y_{i,t} = 0$.
- Constraint (4): if $x_{i,t} = 0$ then $s_{i,t} = 0$.
- Constraint (5): the start time for a sub-job in a window must be within 2 break-times.
- Constraints (6): sub-jobs must not overlap within a window, passed from the following condition:

$$\left[\begin{array}{l} c_{i,t} \leq s_{j,t}; \forall i, j = 1, \ldots, n | i \neq j; \forall t = 1, \ldots, m \\ c_{j,t} \leq s_{i,t}; \forall i, j = 1, \ldots, n | i \neq j; \forall t = 1, \ldots, m \end{array}\right.$$

- Constraints (7): the completion time for a job must not exceed the deadline for this job.

Algorithm 1: FCFS, with $O(m \times n)$

input: *Jobs*: list jobs,
Wins: list windows

begin
  foreach *window* $W_t \in Wins$ **do**
    foreach *job* $J_i \in Jobs$ **do**
      Assignment(J_i, W_t);
    end
  end
end

Algorithm 2: Assignment, with $O(1)$

input: J_i: the i^{th} job,
W_t: the t^{th} window

begin
  if $rj_i \geq split_{\min}$ ***and*** $rw_t \geq split_{\min}$ **then**
    if $rj_i \leq rw_t$ **then**
      Assign job J_i with the size of rj_i into window W_t;
    else
      if $rj_i - rw_t \geq split_{\min}$ **then**
        Assign job J_i with the size of rw_t into window W_t;
      else if $rj_i - split_{\min} \geq split_{\min}$ **then**
        Assign job J_i with the size of $(rj_i - split_{\min})$ into window W_t;
    end
  end
end

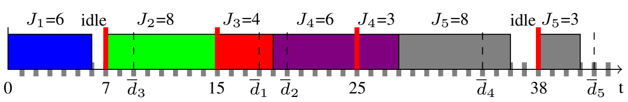


Figure 5. The FCFS with infeasible solution.

3 Proposed Approaches

3.1 Heuristics

3.1.1 First Come First Served (FCFS): The idea of this heuristic is to apply the FCFS rule (the job that comes first will be processed first). It browses each window from LEFT to RIGHT, at each window the assignment of jobs to the window will be considered in one of the following three cases:

- if $rj_i \leq rw_t$ then assign job J_i with size rj_i to window W_t.
- if $rj_i \geq (rw_t + split_{\min})$, then assign job J_i of size rw_t to the window W_t, the rest is put back into the list of jobs.
- if $rj_i \geq (2 \times split_{\min})$, then assign job J_i of size $(rj_i - split_{\min})$ to window W_t, the rest is returned to the list of jobs.

A comment that applying the FCFS rule very easily leads to a job that violates the constraint (7) and will be an infeasible solution. The solution from the FCFS with the above input data in Table I, II presents as Figure 5,

Algorithm 3: EDL, with $O(m \times n \times logn)$

input: *Jobs*: list jobs,
Wins: list windows

begin
  foreach *window* $W_t \in Wins$ **do**
    List jobs sorted in EDL order;
    foreach *job* $J_i \in Jobs$ **do**
      Assignment(J_i, W_t);
    end
  end
end

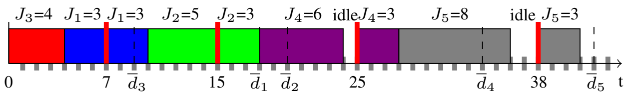


Figure 6. The EDL with $C_{\max} = 41$.

where J_3 violated the constraint (7) so the solution is infeasible.

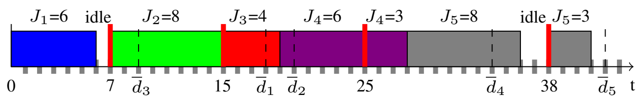
3.1.2 Earliest Deadline (EDL): The idea of this heuristic is to apply the EDL rule (the job with the earliest deadline will be prioritized for processing first). It browses each window from LEFT to RIGHT, at each window the jobs are sorted in incrementing deadline order, then assigning jobs to the window is similar to the FCFS algorithm.

A comment that applying the EDL rule will help to limit the jobs that violate the constraint (7). The solution from the EDL with the above input data in Table I, II presents as Figure 6.

3.2 Neighborhood Search

Note in the FCFS presented at 3.1.1, the order of jobs will affect the solution obtained. For example with the above input data in Table I, II.

Jobs: $\{J_1 = 6, J_2 = 8, J_3 = 4, J_4 = 9, J_5 = 11\}$
$\Rightarrow$ infeasible solution



Jobs: $\{J_3 = 4, J_1 = 6, J_2 = 8, J_4 = 9, J_5 = 11\}$
$\Rightarrow$ feasible solution with $C_{max} = 41$

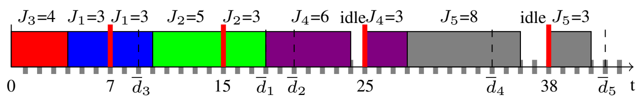


Figure 7. The obtained solutions are different when the order of jobs changes.

This property leads to trying to change the order of jobs to find a better solution that will bring this problem into the form of a combinatorial optimization problem with the fitness value as the $C_{\max}$. Several methods of neighborhood search algorithms such as hill

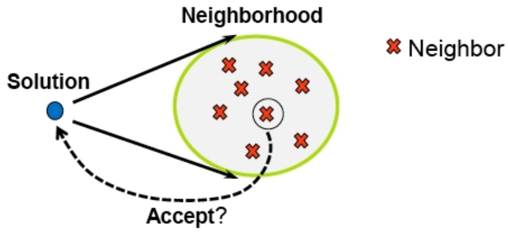


Figure 8. Neighborhood search.

climbing, simulated annealing, tabu search,... and evolutionary strategies such as genetic algorithm, memetic algorithm,... have been proposed to solve combinatorial optimization problems with better solutions. However these methods must be executed through many iterations to find a better solution than the original solution, so the cost in terms of time will increase many times compared to the heuristics. Neighborhood search is a technique used to find the best possible solution to a problem, this technique does not guarantee to find the optimal solution but guarantees to find the best possible solution, based on the method of searching for neighborhood until the stopping condition is satisfied.

The problem is encoded so that neighborhood search can be applied as follows:

- Each solution will represent a set of ordered jobs, for example $S = J_2|J_3|J_5|J_4|J_1$ will be different from $S' = J_3|J_2|J_4|J_1|J_5$.
- Neighbor solution is generated the way as Sourd [4] and Ta [5], which is from current solution $S = J_1|J_i|J_2|J_j|J_3$ the following operations:
  - SWAP: swap two jobs J_i and J_j $\Longrightarrow S' = J_1|J_j|J_2|J_i|J_3$.
  - EBSR (extraction and backward-shifted reinsertion): extract job J_j and insert it before job J_i $\Longrightarrow S' = J_1|J_j|J_i|J_2|J_3$.
  - EFSR (extraction and forward-shifted reinsertion): extract job J_i and insert it after job J_j $\Longrightarrow S' = J_1|J_2|J_j|J_i|J_3$.
  - ALL: combines all 3 operations SWAP, EBSR and EFSR.
- The fitness value of the solutions is the $C_{\max}$ value found in the FCFS algorithm.

3.2.1 Stochastic Hill Climbing (SHC): Stochastic Hill Climbing is a variation of the Basic Hill Climbing method described in [6].

The idea of the SHC algorithm is that while Basic Hill Climbing method tries to find the highest slope in the neighborhood to move to, Stochastic Hill Climbing tries to find a random higher slope in the neighborhood to move to [7]. At each iteration, generate a neighbor solution *newSol* (by operations like SWAP or EBSR or EFSR or ALL) and compare with current solution *currSol*, if *newSol* is better than *currSol* then update *currSol*.

A comment that the Basic Hill Climbing or Stochastic Hill climbing method is quite simple, so the solutions often fall into the local optimum (see Figure 10).

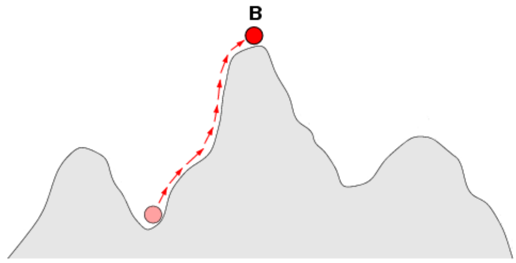


Figure 9. Hill Climbing.

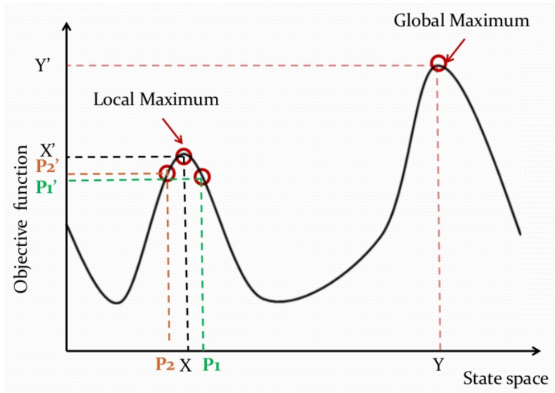


Figure 10. Local optimum vs. Global optimum.

Algorithm 4: SHC

input: *currSol*: current solution,
$iter_{\max}$: maximum number of iterations

begin
  $iter = 1$;
  while $iter \le iter_{\max}$ **do**
    $newSol$ = Neighbor($currSol$);
    if $fitness(newSol) < fitness(currSol)$ **then**
      $currSol = newSol$;
      $iter = 1$;
    else
      $iter = iter + 1$;
    end
  end
  return $currSol$;
end

Algorithm 5: Neighbor

input: *currSol*: current solution

begin
  create randomly i, j positions ($i < j$) in the current solution;
  $newSol$ = SWAP($currSol, i, j$) or EBSR($currSol, i, j$) or EFSR($currSol, i, j$) or ALL($currSol, i, j$);
  return $newSol$;
end

3.2.2 Random Restart Hill Climbing (RRHC): The idea of the RRHC algorithm is that to get out of the local optimal point, it performs a series of hill-climbing searches from different randomly generated initial

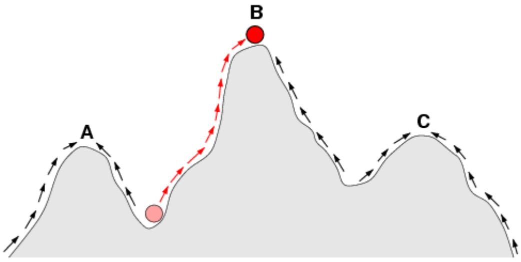


Figure 11. Random Restart Hill Climbing.

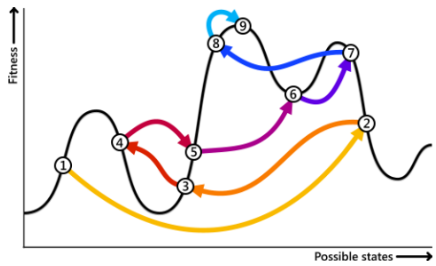


Figure 12. Simulated Annealing.

Algorithm 6: RRHC

```
input: restart_max: maximum number of restarts,
       iter_max: maximum number of iterations
begin
    restart = 1;
    while restart ≤ restart_max do
        initSol = create a randomly initial
          solution;
        localOptSol = SHC(initSol, iter_max);
        if fitness(localOptSol) < fitness(bestSol)
          then
            bestSol = localOptSol;
        end
        restart = restart + 1;
    end
    return bestSol;
end
```

states [7]. At each restart step, generate random initial solution, use SHC algorithm to find local optimum solution *localOptSol* and compare with best solution *bestSol*, if *localOptSol* is better than *bestSol* then update *bestSol* again.

3.2.3 Simulated Annealing (SA): The idea of the SA algorithm is to avoid local optimization by accepting a worse solution with a temperature dependent probability T [8]. At each temperature T, create a neighbor solution *newSol* and evaluate with current solution *currSol*, if *newSol* is better then update *currSol*, otherwise, it can still be accepted with some probability.

Algorithm 7: SA

```
input: Tmax: initial temperature,
       Tmin: finish temperature,
       alpha: cooling rate
begin
    currSol = initSol;
    T = Tmax;
    while T > Tmin do
        newSol = Neighbor(currSol);
        ΔE = fitness(newSol) − fitness(currSol);
        if min(1, e^(−ΔE/T)) ≥ rand(0, 1) then
            currSol = newSol;
        end
        if fitness(currSol) < fitness(bestSol) then
            bestSol = currSol;
        end
        T = alpha * T;
    end
    return bestSol;
end
```

4 Experimental Results

4.1 Dataset

There are two datasets DS1 and DS2 created to evaluate the methods in Section 3. Therein, DS1 contains small sample sizes of 10 to 30 jobs used to compare the exact method with heuristics and neighborhood searches; and DS2 contains large sample sizes of 100 to 300 jobs (scaled up to 10 times) used for comparison between heuristics and neighborhood searches. In each dataset, there are 15 tuples $(n, split_{\min})$ created according to the following rules:

- Each job can be split but not too small, so $split_{\min} \in \{2, 3, 4\}$.
- Normally each job has a maximum processing time of 24h, so p_i is randomly generated by integer uniform distribution in $[split_{\min}, 24]$.
- For each job, a due-date value and a deadline value are generated in the same way as Hariri & Potts [9] and Baptiste [10]:
  - d_i is randomly generated by integer uniform distribution in $[d^l P, d^u P]$, where $P = \sum_{i=1}^{n} p_i$, $d^l \in \{0.1, 0.3, 0.5, 0.7\}$, $d^u \in \{0.3, 0.5, 0.7, 0.9\}$, $d^l < d^u$.
  - $\bar{d}_i$ is randomly generated by integer uniform distribution in $[d_i, 1.1 \times P]$.

  Thus, in each tuple, there will be 10 instances created with 10 sets of values d^l and d^u.
- According to the International Labor Office - Geneva [11], the working time in a day does not exceed 12h, and a window of time corresponding to a day can be viewed, so w_t is randomly generated by integer uniform distribution in $[split_{\min}, 12]$ and the number of windows m such that $\sum_{t=1}^{m} w_t \geq \sum_{i=1}^{n} p_i$.

For an example with a tuple (10, 4), the input data of 10 instances are created as Table III.

The Figure 13 shows the density distribution of the data $p_i, \bar{d}_i, w_t$ for the tuple $(n = 10, split_{\min} = 4, d^l = 0.1, d^u = 0.3)$.

Table III
The Input Data Generated with a Tuple (10,4)

ID	d^l	d^u	p_i = 11 15 11 20 12 16 16 12 6 8 w_t = 10 4 5 7 6 8 5 4 10 6 8 9 7 10 10 8 11
1	0.1	0.3	$\overline{d}_i$ = 107 72 56 91 71 76 111 76 84 89
2	0.1	0.5	$\overline{d}_i$ = 111 85 54 128 87 61 60 85 68 51
3	0.1	0.7	$\overline{d}_i$ = 106 137 61 130 78 126 101 116 35 66
4	0.1	0.9	$\overline{d}_i$ = 60 69 120 97 62 114 63 46 122 34
5	0.3	0.5	$\overline{d}_i$ = 119 101 75 122 134 57 39 101 83 136
6	0.3	0.7	$\overline{d}_i$ = 108 96 127 40 119 97 104 49 78 101
7	0.3	0.9	$\overline{d}_i$ = 117 101 126 132 128 114 62 125 113 75
8	0.5	0.7	$\overline{d}_i$ = 101 134 106 135 113 82 110 73 119 96
9	0.5	0.9	$\overline{d}_i$ = 95 137 117 136 115 100 121 135 114 72
10	0.7	0.9	$\overline{d}_i$ = 139 113 127 119 137 124 104 118 135 119

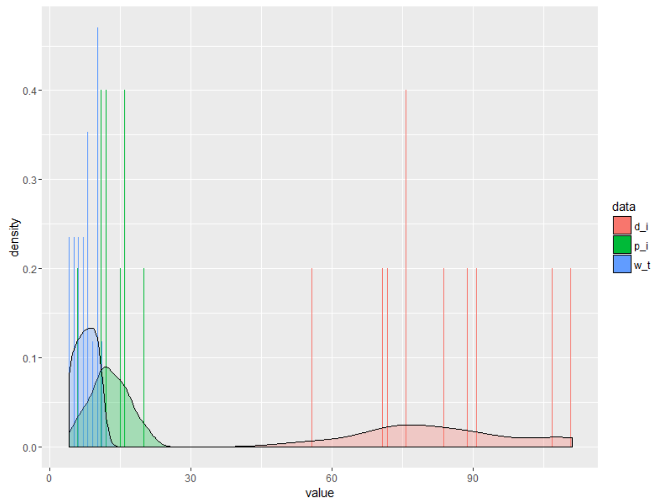


Figure 13. The density distribution of data for the tuple ($n = 10, split_{\min} = 4, d^l = 0.1, d^u = 0.3$)

4.2 Evaluation Criteria

All relevant algorithms were experimentally installed on a computer configured with Intel(R) Core(TM) i7-4650U 1.70GHz, 8GB memory with Windows 8.1 Professional OS, with three evaluation criteria:

- The number of feasible solutions found (#FS).
- If the solution is feasible then the percentage gap (%LB) between the $C_{\max}$ value found and the lower bound $LB = \sum_{i=1}^{n} p_i$.
- The runtime (t) found the solution.

Besides, the exact method with solving the MILP model is also installed on CPLEX 12.7.1 solver to find the number of solutions (#FS) that are not only feasible solutions but also optimal solutions and percentage gap (%LB) between the optimal solution and LB.

4.3 Configuration Parameters

The configuration parameters in the algorithms greatly affect the quality of the solution found as well as the processing time of the algorithm. For example for the SHC algorithm you have to choose which operations to generate neighbor solutions, while for the RRHC algorithm how many iterations will be reasonable, and finally, the SA algorithm will what should be the initial temperature as well as the heat reduction coefficient.

4.3.1 SHC operators: In the SHC algorithm, choosing how to create the neighbor solution is very important. The four proposed operations that are SWAP, EBSR, EFSR, and ALL will be experimented on the dataset DS1 to find out suitable operations for creating the neighbor solution.

Table IV shows that the ALL operation resulted in the highest number of feasible solutions found (107 solutions), as well as the percentage gap (%LB) between the results $C_{\max}$ and the lower bound LB is the lowest (2.46%). Therefore, choose the combined operation ALL to implement and test algorithms such as RRHC and SA.

4.3.2 RRHC parameters: In the RRHC algorithm, the quality of the solution will be improved after each iteration. So the question is how many iterations are needed? Does too much iteration affect the processing time of the algorithm? We conduct experiments on DS1 to find the right value for two parameters $restart_{\max}$ and $iter_{\max}$. To ensure that all instances give a feasible solution, select instances generated by the tuple ($d^l = 0.7, d^u = 0.9$) because with this tuple the deadline value is generated within the widest possible normal distribution.

Table V shows that the higher the number of iterations, the higher the processing time of the algorithm. And the set of parameter values ($restart_{\max}$=1000 and $iter_{\max}$=100) gives the lowest average value of $C_{\max}$ (231.87) and the average processing time is 31.08 seconds is acceptable. Therefore, this parameter set is for the comparative evaluation of the algorithms in 4.4. The convergence of the hill climbing is shown in Figure 14 and it also shows how the best solution improves after restarts.

4.3.3 SA parameters: In the SA algorithm, the value of initial temperature (Tmax) and the cooling factor (alpha) will affect the quality of the algorithm's solution, because these parameter values will change to the current temperature value at each iteration, thereby affecting the acceptance probability during execution.

Table IV
THE SHC OPERATORS ON THE DS1

ID	*n*	$split_{min}$	SWAP			EBSR			EFSR			ALL		
			#FS	%LB	t	#FS	%LB	t	#FS	%LB	t	#FS	%LB	t
1	10	2	8	1.42	0.03	8	0.77	0.03	8	1.16	0.04	8	0.52	0.06
2	10	3	7	1.44	0.03	7	1.44	0.03	7	1.64	0.03	7	1.44	0.07
3	10	4	4	4.13	0.03	4	4.13	0.03	4	4.92	0.03	4	4.13	0.06
4	15	2	6	0.96	0.07	7	0.73	0.07	6	0.85	0.07	7	0.64	0.13
5	15	3	7	2.23	0.06	7	1.95	0.06	7	2.5	0.06	7	1.67	0.11
6	15	4	6	5.13	0.07	6	4.34	0.07	6	5.13	0.07	6	4.03	0.14
7	20	2	7	0.77	0.11	7	0.64	0.11	7	0.9	0.11	7	0.58	0.21
8	20	3	8	3.53	0.1	8	3.35	0.09	8	2.69	0.1	8	2.39	0.21
9	20	4	5	5.49	0.1	5	5.49	0.1	5	5.91	0.1	5	4.98	0.22
10	25	2	9	0.89	0.17	9	0.43	0.17	9	1.01	0.18	9	0.7	0.33
11	25	3	7	3.13	0.18	7	3.23	0.16	7	3.56	0.19	7	2.37	0.31
12	25	4	5	5.54	0.15	6	5.73	0.15	5	5.75	0.14	6	4.68	0.32
13	30	2	10	0.99	0.28	10	0.51	0.29	10	0.9	0.28	10	0.88	0.51
14	30	3	9	3.19	0.27	9	2.93	0.27	9	3.55	0.26	9	2.7	0.47
15	30	4	7	6.5	0.25	7	4.67	0.25	7	6.72	0.25	7	5.13	0.44
Total			105	-	-	107	-	-	105	-	-	107	-	-
Average			-	3.02	-	-	2.69	-	-	3.15	-	-	2.46	-

Notes:

- (#FS): higher is better; (%LB) and (t): lower is better
- Each tuple $(n, split_{min})$ is the average results of 10 sample instances

Table V
SUMMARY OF RESULTS WITH PARAMETERS $restart_{max}$ AND $iter_{max}$

nr	ni	(10,2)	(10,3)	(10,4)	(15,2)	(15,3)	(15,4)	(20,2)	(20,3)	(20,4)	(25,2)	(25,3)	(25,4)	(30,2)	(30,3)	(30,4)	average	runtime
10	10	97	140	132	158	156	218	224	211	245	287	271	302	355	343	392	235.40	0.04
10	100	97	140	132	157	154	218	224	211	244	287	271	300	354	343	389	234.73	0.31
10	1000	97	140	132	157	154	218	224	210	244	287	267	296	354	343	388	234.07	2.55
100	10	97	140	128	157	154	218	224	211	245	287	267	300	354	342	388	234.13	0.34
100	100	97	140	128	157	154	214	224	209	240	287	267	294	354	342	383	232.67	3.11
100	1000	97	140	128	157	154	214	224	209	240	287	265	292	354	342	384	232.47	25.44
1000	10	97	140	128	157	154	218	224	210	240	287	267	296	354	343	388	233.53	3.33
1000	100	97	140	128	157	154	214	224	209	240	287	266	290	354	339	379	231.87	31.08
1000	1000	97	140	128	157	154	214	224	209	240	287	265	290	354	339	381	231.93	257.36

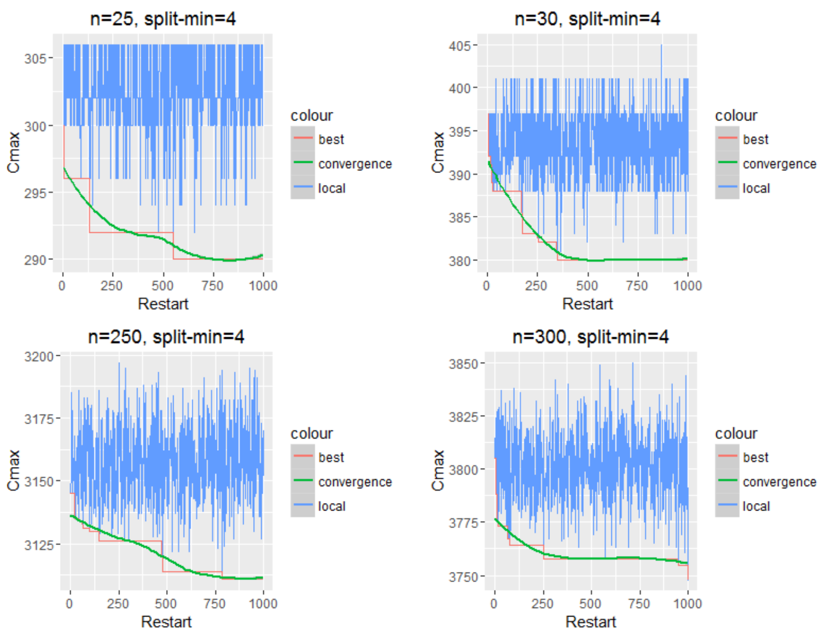


Figure 14. The convergence of the RRHC algorithm on several instances.

Table VI
Summary of Results with $Tmax = 10000$

alpha	(10,2)	(10,3)	(10,4)	(15,2)	(15,3)	(15,4)	(20,2)	(20,3)	(20,4)	(25,2)	(25,3)	(25,4)	(30,2)	(30,3)	(30,4)	average
0.99	97	140	132	158	154	218	224	214	246	287	271	296	356	346	392	235.40
0.991	97	140	132	158	156	219	224	211	240	288	273	296	355	343	391	234.87
0.992	97	140	132	158	156	220	224	213	245	287	271	300	355	345	388	235.40
0.993	97	140	128	158	156	214	224	211	244	287	271	294	355	344	392	234.33
0.994	97	140	128	158	154	218	224	212	245	287	270	300	355	345	388	234.73
0.995	97	140	128	158	154	219	224	210	244	287	271	296	355	347	391	234.73
0.996	97	140	132	157	154	218	224	211	245	288	271	294	355	345	388	234.60
0.997	97	140	132	158	154	218	224	213	244	287	271	294	354	345	388	234.60
0.998	97	140	128	157	156	214	224	210	241	287	266	290	354	343	388	233.00
0.999	97	140	128	157	154	214	224	211	241	287	267	288	354	343	388	232.87

Table VII
Summary of Results with $alpha = 0.999$

Tmax	(10,2)	(10,3)	(10,4)	(15,2)	(15,3)	(15,4)	(20,2)	(20,3)	(20,4)	(25,2)	(25,3)	(25,4)	(30,2)	(30,3)	(30,4)	average
1000	97	140	128	157	154	214	224	210	241	287	267	291	354	343	388	233.00
10000	97	140	128	157	154	214	224	209	241	287	267	290	354	343	384	232.60
100000	97	140	128	157	154	214	224	210	245	287	273	306	355	343	401	235.60
1000000	97	140	128	157	154	214	224	211	244	287	267	306	354	342	388	234.20
10000000	97	140	128	158	154	214	224	211	241	287	277	306	354	345	400	235.73

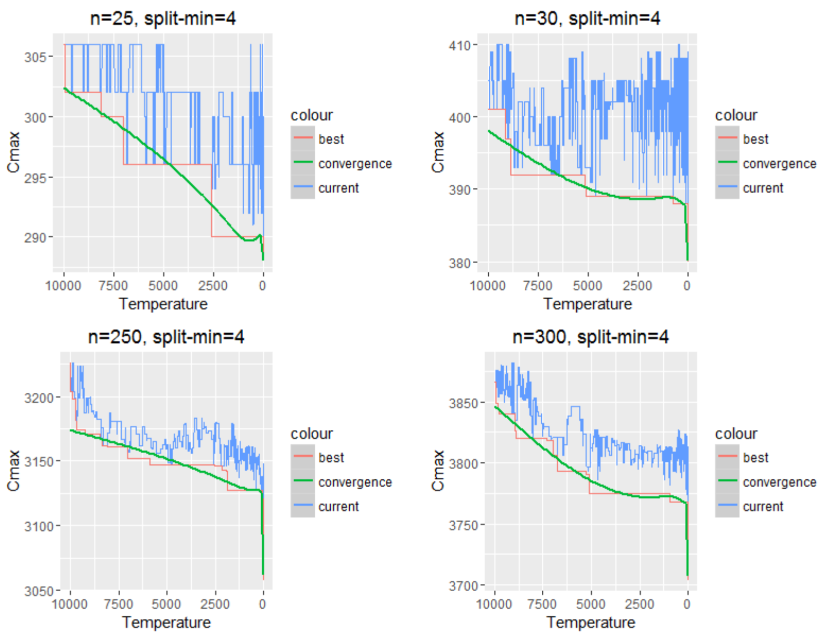


Figure 15. The convergence of SA algorithm on several instances.

According to Laarhoven [12], the parameter value of $Tmax$ should be very high and the value of $alpha$ should be close to 1 for slow temperature reduction. First, select $Tmax = 10000$, conduct experiments on the DS1 to find the value $alpha \in \{0.990, 0.991, 0.992, 0.993, 0.994, 0.995, 0.996, 0.997, 0.998, 0.999\}$ is suitable for the problem. To ensure that all instances give a feasible solution, select instances are generated by the tuple $(d^l = 0.7, d^u = 0.9)$ because with this tuple the deadline value is generated within the widest possible normal distribution.

Table VI shows that $alpha = 0.999$ achieves the minimum C_{max} mean (232.87). Therefore, choose $alpha = 0.999$, continue to experiment on the DS1 to find the value $Tmax \in \{1000, 10000, 100000, 1000000, 10000000\}$. Table VII shows that $Tmax = 10000$ achieves the minimum C_{max} (232.60). Therefore, choose $alpha = 0.999$ and $Tmax = 10000$ for the comparison evaluation of algorithms in 4.4 section. The convergence of the simulated annealing process is shown in Figure 15 and it also shows how the best solution improves during the process from $Tmax$ to $Tmin$.

Table VIII
Summary of Results on DS1

ID	*n*	$split_{\min}$	CPLEX			EDL			SHC			RRHC			SA		
			#FS	%LB	t	#FS	%LB	t	#FS	%LB	t	#FS	%LB	t	#FS	%LB	t
1	10	2	8	0	0.53	8	3.35	0	8	0.52	0.06	8	0.39	0.65	8	0.39	0.93
2	10	3	8	0.72	0.63	6	3.36	0	7	1.44	0.07	7	0.72	0.79	7	0.72	1.1
3	10	4	6	0	2.76	2	5.51	0	4	4.13	0.06	5	2.68	0.67	6	1.84	0.93
4	15	2	7	0.09	6.34	6	2.87	0	7	0.64	0.13	7	0.27	1.48	6	0.21	2.03
5	15	3	8	0	3.77	5	5.84	0	7	1.67	0.11	7	0	1.43	7	0	1.69
6	15	4	9	1.42	26.02	3	6.32	0	6	4.03	0.14	6	3	1.74	6	1.82	2.1
7	20	2	7	0.45	11.51	7	2.05	0	7	0.58	0.21	7	0.45	2.45	7	0.45	3.7
8	20	3	9	0	38.12	4	5.86	0	8	2.39	0.21	9	0.64	2.24	4	0.12	2.82
9	20	4	7	1.27	121.43	4	6.65	0	5	4.98	0.22	6	1.83	2.52	6	1.27	3.28
10	25	2	9	0	34.73	9	2.4	0	9	0.7	0.33	9	0	4.35	9	0	5.08
11	25	3	8	0	72.99	7	5.98	0	7	2.37	0.31	8	0.99	4.07	7	0.16	4.33
12	25	4	8	1.05	319.38	1	7.37	0	6	4.68	0.32	7	2.61	3.17	1	1.05	3.66
13	30	2	10	0	115.82	10	2.49	0	10	0.88	0.51	10	0.03	6.76	10	0	7.58
14	30	3	9	0	283.86	7	5.71	0	9	2.7	0.47	9	1.31	7.07	7	0.3	6.58
15	30	4	8	0	1248.26	4	8.11	0	7	5.13	0.44	8	3.13	5.99	4	1.26	6.11
Total			121	-	-	83	-	-	107	-	-	113	-	-	95	-	-
Average			-	0.33	152.41	-	4.92	0.00	-	2.46	0.24	-	1.2	3.03	-	0.64	3.46

Notes:
- (#FS): higher is better; (%LB) and (t): lower is better
- Each tuple $(n, split_{\min})$ is the average results of 10 sample instances

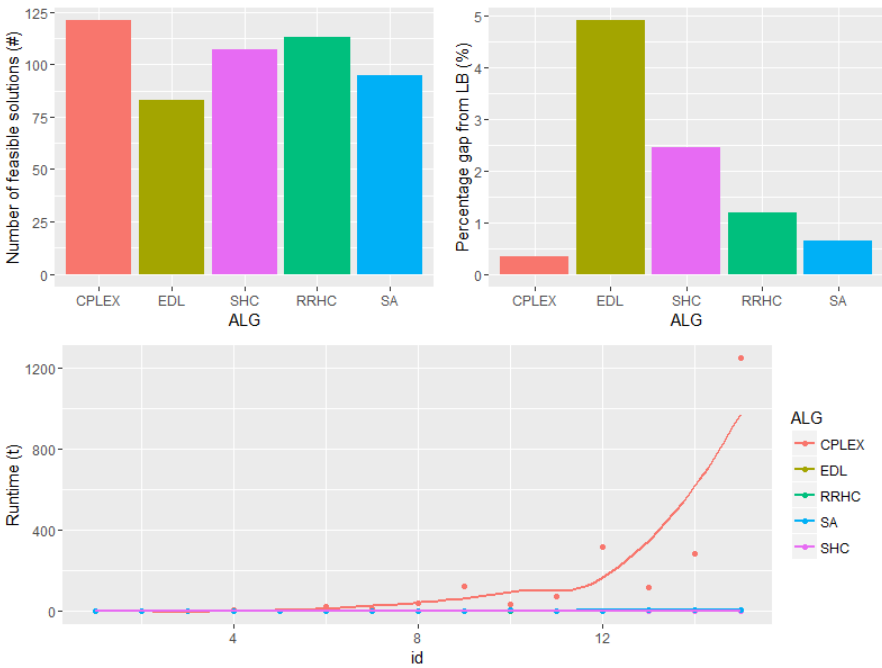


Figure 16. The comparison chart on DS1.

4.4 Benchmarking

The first experiment on the DS1, the results are summarized in Table VIII and Figure 16. Experimental results show that the CPLEX solver finds more feasible solutions than all other algorithms (121 solutions), in addition, when finding feasible solutions, the percentage gap (%LB) between results $C_{\max}$ is found and the lower bound *LB* is also the smallest (0.33%). However, the time for the CPLEX solver to find the solution is very high (the case $n = 30$ and $split_{\min} = 4$ takes more than 20 minutes) and increases exponentially with n. In contrast, the EDL heuristic has a very fast running time (0.00 seconds for all cases) but the quality of the solutions is not good, the number of feasible solutions found in the lowest (83 solutions) and the percentage gap (%LB) is the highest (about 4.92%). In the Hill Climbing algorithms, the RRHC algorithm gives better solution quality than the SHC algorithm with more feasible solutions (113 versus 107 solutions) and a lower

Table IX
Summary of Results on DS2

ID	*n*	$split_{\min}$	EDL			SHC			RRHC			SA		
			#FS	%LB	t	#FS	%LB	t	#FS	%LB	t	#FS	%LB	t
1	100	2	10	2.32	0	10	1.11	4.23	10	0.73	57.22	10	0.2	65.82
2	100	3	10	5.51	0	10	3.14	6.22	10	2.54	60.64	10	1.13	57.41
3	100	4	4	9.44	0	8	6.16	4.89	9	5.68	47.79	4	2.44	52.31
4	150	2	10	2.48	0.01	10	1.23	12.17	10	0.96	139.11	10	0.41	156.31
5	150	3	10	5.27	0	10	3.39	14.09	10	2.92	148.35	10	1.62	136.96
6	150	4	6	8.96	0	10	5.66	16.55	10	5.52	196.31	6	3.04	130.51
7	200	2	10	2.21	0.01	10	1.27	23.13	10	1.02	280.41	10	0.63	285.62
8	200	3	10	5.46	0.01	10	3.72	23.5	10	3.31	253.2	10	1.92	241.57
9	200	4	4	9.36	0.01	9	6.56	31.48	9	6.15	229.28	4	3.04	218.08
10	250	2	10	2.26	0.01	10	1.42	40.6	10	1.16	402.07	10	0.6	390.54
11	250	3	10	5.87	0.01	10	3.95	48.44	10	3.54	453.65	10	1.96	333.18
12	250	4	5	9.14	0.02	10	6.57	53.31	10	6.69	387.15	6	3.77	313.86
13	300	2	10	2.57	0.02	10	1.65	64.48	10	1.45	540.98	10	0.87	534.75
14	300	3	10	5.78	0.02	10	3.93	80.6	10	3.8	608.76	10	2.3	552.81
15	300	4	6	9.1	0.02	9	6.96	85.56	9	6.71	539.91	8	3.93	471.79
Total			125	-	-	146	-	-	147	-	-	128	-	-
Average			-	5.72	0.01	-	3.78	33.95	-	3.48	289.66	-	1.86	262.77

Notes:
- (#FS): higher is better; (%LB) and (t): lower is better
- Each tuple $(n, split_{\min})$ is the average results of 10 sample instances

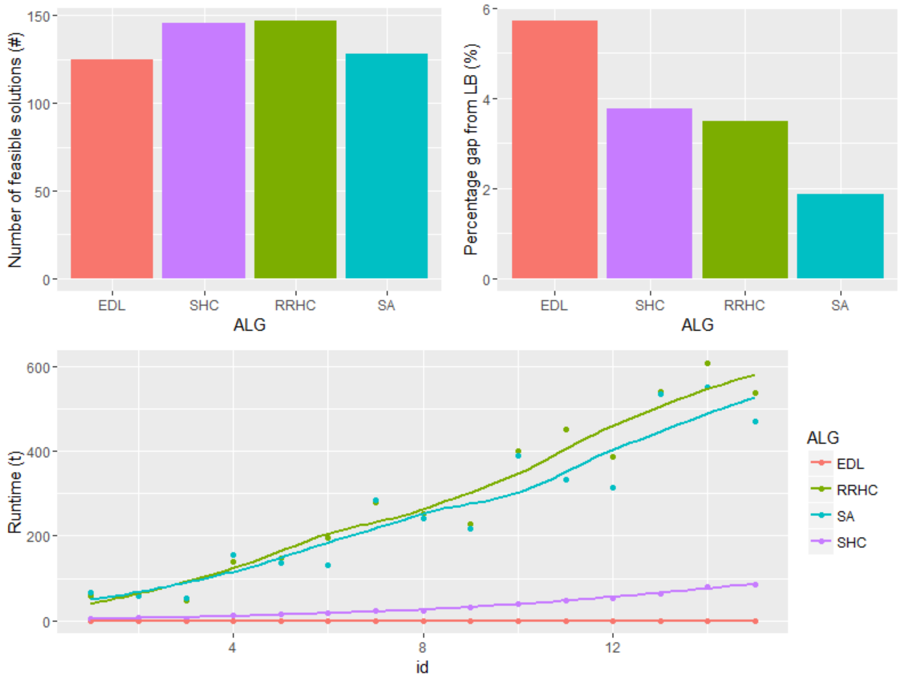


Figure 17. The comparison chart on DS2.

percentage gap (%LB) (1.2% compared to 2.46%), but the processing time of the RRHC is many times higher than that of the SHC algorithm because the RRHC algorithm has to restart many times. Besides, the RRHC algorithm also finds the most feasible solution among all approximation algorithms (113 solutions). Finally, the SA algorithm gives a compromise solution with good solution quality (#FS=95 and %LB=0.64) while the running time is low (just over 3 seconds).

The next experiment on the large DS2, the results are summarized in Table IX and Figure 17. This experiment only compares the results of heuristic and neighborhood searches without the CPLEX solver. Because the CPLEX solver not only finds a feasible solution but also tries to find the optimal solution, so for large data sets the CPLEX solver runs very long compared to the acceptable time of the problem is about 10 minutes. The experimental results show that similar to the DS1, the EDL algorithm finds the solution very quickly (only about 0.01 seconds), but the quality of the solution is not good with the percentage gap (%LB) is quite high (about 5.72%). The Hill Climbing algorithms (SHC and RRHC) found the highest number of feasible solutions (146 and 147 solutions), while the SA algorithm had the lowest percentage gap (%LB) (only about 1.86%). In terms of processing time, the RRHC algorithm and the SA algorithm have a fairly high average processing time (about 5 minutes), especially in the case ($n = 300$ and $split_{\min} = 3$) it takes about 10 minutes to process, which is understandable since RRHC and SA both have strategies to get rid of the local optimal solution by iterating many times in the hope that the global optimal solution can be reached. Another note for the cases of $split_{\min} = 4$, the number of infeasible solutions is quite high because in this case, the jobs are difficult to break down to assigned into the appropriate windows.

In summary, with the number of jobs $n \leq 30$, we should use the CPLEX solver to determine the optimal solution to the problem in an acceptable time (under 10 minutes). In contrast, with a larger number of jobs, if the priority is given to the criterion of finding the number of feasible solutions as high as possible, then the RRHC algorithm should be chosen, and if the priority is for the percentage gap (%LB) criterion the more as low as possible, the SA algorithm should be used.

5 Conclusion

In this paper, the problem of scheduling individual jobs so that all jobs are completed at the earliest within time windows with constraints $split_{\min}$ and deadlines have been set and solved. The MILP model has been built and implemented using the CPLEX solver to determine the optimal solution to the problem. In addition, several heuristics such as FCFS, EDL, and neighborhood searches such as SHC, RRHC, SA have been proposed to determine the feasible solution for this problem. Experiments to evaluate the proposed methods have also been performed and the results show that the RRHC and SA algorithms achieve a compromise between good solution quality and acceptable execution time in both small and large sample sizes datasets. Adding more constraints to this personal scheduling problem is an issue to consider in the future, such as constraints on the order of jobs or constraints on parallel machines.

References

[1] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[2] V. Nguyen, N. H. Tuong, H. Nguyen, and T. Nguyen, "Single-machine scheduling with splitable jobs and availability constraints," *REV Journal on Electronics and Communications*, vol. 3, no. 1-2, pp. 21–27, 2013.

[3] V. Nguyen, N. H. Tuong, V. Tran, and N. Thoai, "An MILP-based makespan minimization model for single-machine scheduling problem with splitable jobs and availability constraints," in *Proceedings of the International Conference on Computing, Management and Telecommunications (ComManTel)*, Ho Chi Minh, Vietnam, 2013, pp. 397–400.

[4] F. Sourd, "Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints," *Operations Research Letters*, vol. 34, no. 5, pp. 591–598, 2006.

[5] Q. C. Ta, J.-C. Billaut, and J.-L. Bouquard, "Heuristic algorithms to minimize the total tardiness in a flow shop production and outbound distribution scheduling problem," in *Proceedings of the International Conference on Industrial Engineering and Systems Management (IESM)*, Seville, Spain, 21-23 Oct. 2015.

[6] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, 2009, ch. Beyond Classical Search.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[9] A. M. A. Hariri and C. N. Potts, "Single machine scheduling with deadlines to minimize the weighted number of tardy jobs," *Management Science*, vol. 40, no. 12, pp. 1712–1719, 1994.

[10] P. Baptiste, F. Della Croce, A. Grosso, and V. T'Kindt, "Sequencing a single machine with due dates and deadlines: An ILP-based approach to solve very large instances," *Journal of Scheduling*, vol. 13, no. 1, pp. 39–47, 2010.

[11] I. L. O. Geneva, "Working time in the twenty-first century," *International Labour Organization*, 2011.

[12] P. Laarhoven, *Theoretical and Computational Aspects of Simulated Annealing*. Erasmus Universiteit Rotterdam, 1988.

Trang Hong Son is a senior lecturer in the Faculty of Information Technology, Hoa Sen University, Vietnam. He received BSc. degree (2004) in Physics and Computer from HCMC University of Natural Science, Vietnam, and Master of e-Management degree (2010) from Innotech International Academy, France. Now he is a Ph.D. student from HCMC University of Technology, Vietnam.

His previous studies were in areas such as High-Performance Parallel and Distributed Computing, Bioinformatics, and now his research interests include Computational intelligence, Scheduling application, Blockchain technology. Besides his work has appeared in the Journal of Ambient Intelligence and Humanized Computing, the Science & Technology Development Journal - Engineering and Technology, the Journal of Computer Science and Cybernetics.

Tran Van Lang is a senior principal research scientist on computer science of Vietnam Academy of Science and Technology (VAST) who has been interested in the development of bioinformatics and parallel computing in Vietnam. He is an editor-in-chief of the HUFLIT Journal of Science (HJS), journal scientific secretary, and chief editor of the Electronic and Telecommunication Section of the Vietnam Journal Science and Technology.

Before being appointed editor-in-chief of HJS, he was deputy editor-in-chief of the Journal of Computer Science and Cybernetics, dean of the Information Technology Faculty, Lac Hong University, as well as a member of directorship of the Insititute of Applied Mechanics and Informatics, VAST.

He received BSc. degree in mathematics (1982) and Ph.D. Degree in mathematics-physics (1995) from the HCMC University of Natural Sciences (Vietnam). And he has been an Assoc. Professor in computer science from the Graduate University of Science and Technology, VAST (2006). He also has worked as a researcher in computational mathematics at the Dorodnitsyn Computing Center, Russian Academy of Science (Russia).

His interests include bioinformatics, high-performance parallel and distributed computing, computational intelligence, scientific computation, and computational mathematics. His work has appeared in journals such as the BMC Bioinformatics, the Algorithms for Molecular Biology, the Vietnam Journal of Mathematics, the Journal of Computer Science and Cybernetics, Vietnam Journal of Science and Technology,... He also has written a number of books on software development, bioinformatics, grid computing, and elearning. His home page is reached at http://fair.conf.vn/∼lang.

Nguyen Huynh-Tuong is an associate professor in the Faculty of Computer Science and Engineering, HCMC University of Technology, Vietnam. He has received his Ph.D. (2009) and M.Eng (2006) in Computer Science from François Rabelais University (Tours, France), and B.Eng (2001) in Computer Engineering from HCMC University of Technology, Vietnam. His research interests are in the areas of scheduling, high-performance computing, and network security.

He worked for more than ten years as a research expert of algorithms and resolution approaches including simulation, modeling, and optimization for real-life problems, including manufacturing scheduling, transportation problems, education management and assessment, and blockchain applications. His work has appeared in the Asian Journal of Computer Science and Information Technology, the European Journal of Operational Research, the Journal of Scheduling, Mathematical Problems in Engineering, IEEE Access.