Huazhong University of Science and Technology

A Two-Individual Based Evolutionary Algorithm for the Flexible Job Shop Scheduling Problem

Junwen Ding

AAAI-19, Hawaii, USA

January 28, 2019

Outline

Job shop scheduling problem (JSP): Given a set of jobs $J = \{J_1, \ldots, J_n\}$ that must be processed on a set $M = \{M_1, \ldots, M_m\}$ of machines. Each job $J_i, i = 1, \ldots, n$, consists of n_i operations $O_i = \{o_{i1}, \ldots, o_{in_i}\}$ that should be sequentially processed. Besides, each operation o_{ij} requires uninterrupted and exclusive use of its assigned machine for its whole processing time.

- Job set $J = \{J_1, \ldots, J_n\}$.
- Machine set $M = \{M_1, \ldots, M_m\}$.
- Operation set $O_i = \{o_{i1}, \ldots, o_{in_i}\}, i \in \{1, \ldots, n\}$.
- Each operation o_{ij} can be only processed by one machine $m \in M$ and requires a processing time of t_{ij}.

The problem is to assign each operation to a machine and to order the operations on the machines, such that the maximum completion time of all jobs (i.e., makespan) is minimized.

An example

- Three machines M_1, M_2, M_3.
- Three jobs J_1, J_2, J_3.
- Each job has three operations, $J_{1,1}, J_{1,2}, J_{1,3}$; $J_{2,1}, J_{2,2}, J_{2,3}$; $J_{3,1}, J_{3,2}, J_{3,3}$.

Table 1: An instance

Job	Processing sequence (machine, time)		
J_1	(1,3)	(2,2)	(3,3)
J_2	(1,2)	(3,3)	(2,4)
J_3	(2,2)	(1,2)	(3,3)

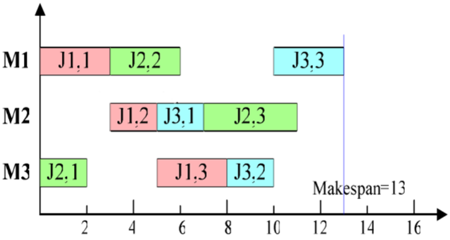


Figure 1: The gantt chat of a solution

The extension of JSP: FJSP

The flexible job shop scheduling problem (FJSP) is an extension of JSP by allowing an operation o_{ij} to be processed on one of a set of candidate machines $M(o_{ij}) \subseteq M$. The processing time of operation o_{ij} on machine $M_k \in M(o_{ij})$ is denoted by $t_{o_{ij}k}$.

- Each operation o_{ij} can be processed on a set of candidate machines $M(o_{ij}) \subseteq M$.
- The processing time of operation o_{ij} on machine $M_k \in M(o_{ij})$ is denoted by $t_{o_{ij}k}$.

Both JSP and FJSP are proven to be NP-hard.

The disjunctive graph is a directed acyclic graph, in which vertices (representing operations to be performed) may be connected by both directed and undirected edges (representing timing constraints between operations).

Table 2: An instance

Job	Processing sequence (machine, time)		
J_1	(1,3)	(2,2)	(3,3)
J_2	(1,2)	(3,3)	(2,4)
J_3	(2,2)	(1,2)	(3,3)

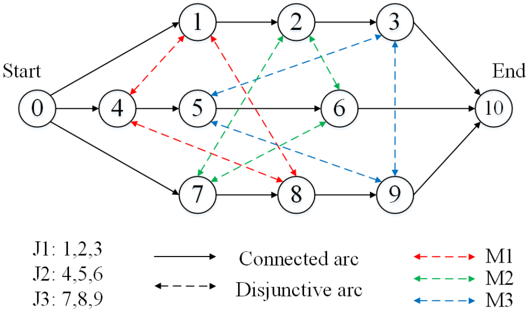


Figure 2: The disjunctive graph of the instance in Table 2

Critical path and operations

Critical path: the longest path from the start node to the end node, of which the length is the makespan.
Critical operations: the operations along the critical path.

When the processing sequence of the operations are determined on each machine, a solution is obtained.

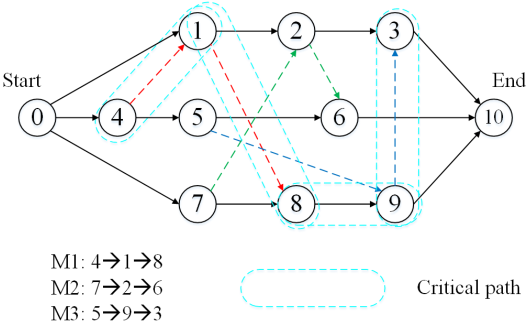


Figure 3: The disjunctive graph of a solution for the instance in Table 2

Critical block

Critical block: The consecutive critical operations on the same machine.

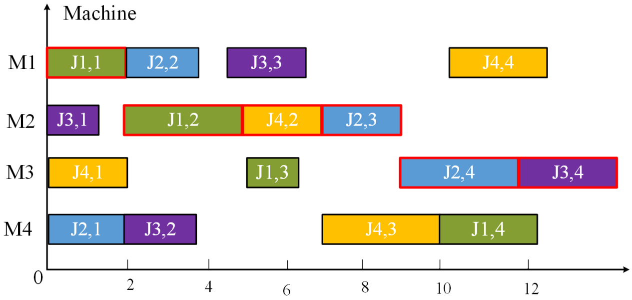


Figure 4: The illustration of critical block

Three critical blocks: (J1,1) (J1,2 J4,2 J2,3) (J2,4 J3,4).

The gantt chart of a solution

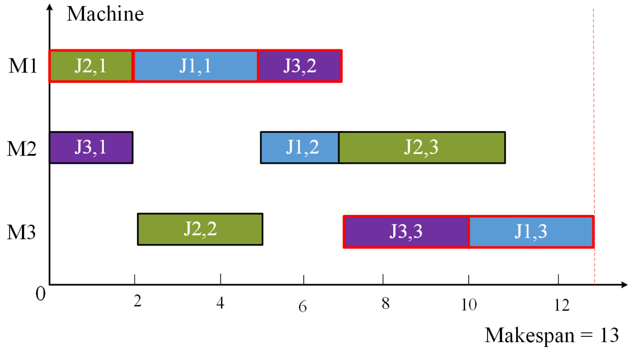


Figure 5: The gantt chart of the solution in Fig. 3

Three types of schedule

- **Active Schedule**: A feasible schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one operation finishing earlier and no operation finishing later.
- **Semi-active Schedule**: A feasible schedule is called semiactive if no operation can be completed earlier without changing the order of processing on any one of the machines.
- **Nondelay Schedule**: A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing (i.e., it prohibits unforced idleness).

The relationship of the schedules

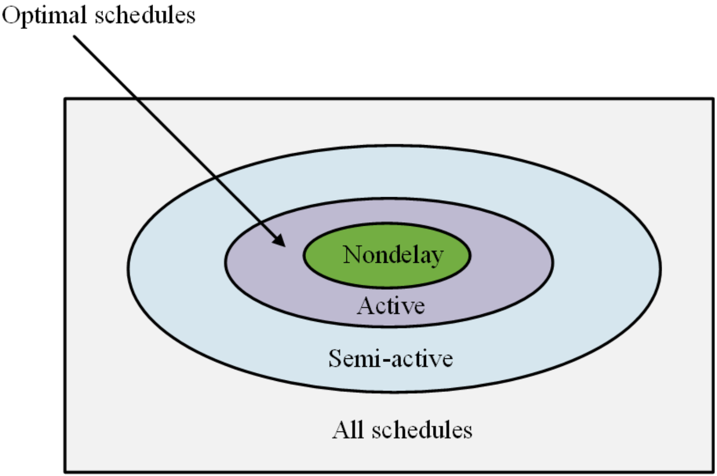


Figure 6: The relationship of the schedules

One example

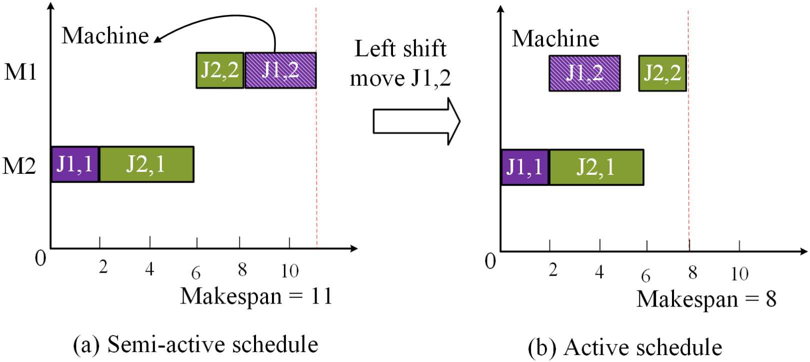


Figure 7: From semi-active schedule to active schedule

Theorems

- Change the sequence of the inner operations of a critical block cannot reduce the makesapn.
- Change the sequence of the non-critical operations in a machine cannot reduce the makesapn.
- Only the change involves the front or the rear operations of a critical block may reduce the makesapn.

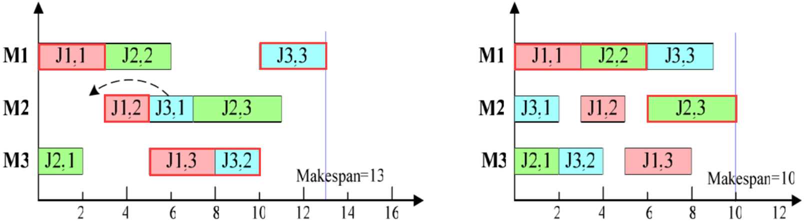


Figure 8: The illustration of reducing the makespan

N5

- N5 was proposed in **?**.
- It only swaps the two consecutive operations on the front and rear of the critical block, in order to avoid moving the inner operations.
- N5 is very fast to search due to the small size, but it is less efficiency because potentiality of searching the promising areas is restricted.

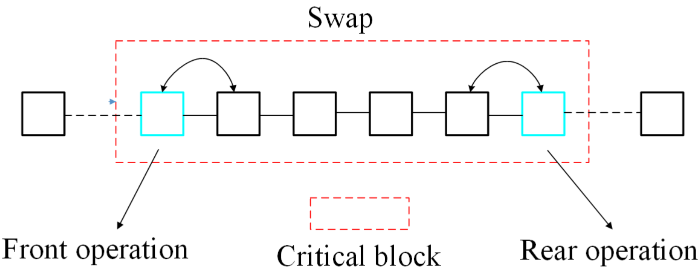


Figure 9: The illustration of N5

N6

- N6 was proposed in **?**.
- It moves the inner operations of a critical block to the front or rear operations of it.

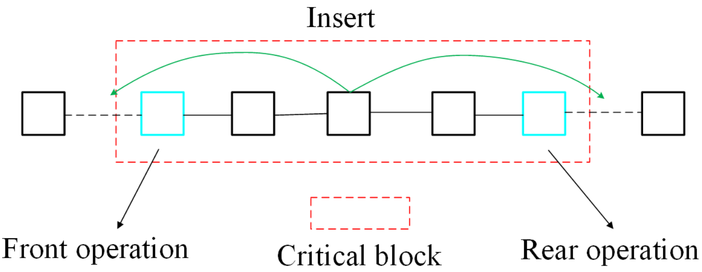


Figure 10: The illustration of N6

N7

- N7 was proposed in Zhang *et al.* [2007].
- Based on N6, N7 additionally moves the front and rear operations of a critical block to the inner positions of it.
- N7 is the most effective neighborhood structure used in local search methods.

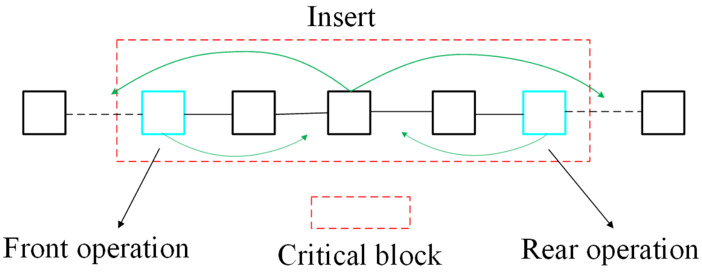


Figure 11: The illustration of N7

Definition of JP, JS, MP, and MS

- $JP[i]$: The predecessor of operation i on the same job.
- $JS[i]$: The successor of operation i on the same job.
- $MP[i]$: The predecessor of operation i on the same machine.
- $MS[i]$: The successor of operation i on the same machine.

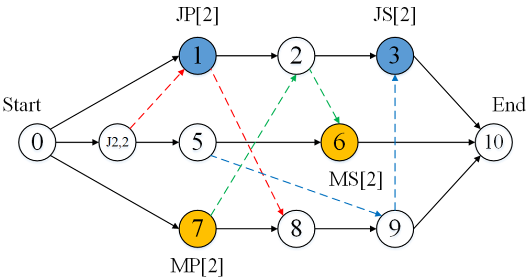


Figure 12: The illustration of JP, JS, MP, and MS

Definition of R and Q

- $R[i]$: The longest path from the start node to operation i.
- $Q[i]$: The longest path from operation i to the end node.
- $R[i] = \max\{R[JP[i]] + t[JP[i]], R[MP[i]] + t[MP[i]]\}$.
- $Q[i] = \max\{Q[JS[i]] + t[JS[i]], Q[MS[i]] + t[MS[i]]\}$.
- For critical operation i, $makespan = R[i] + t[i] + Q[i]$.

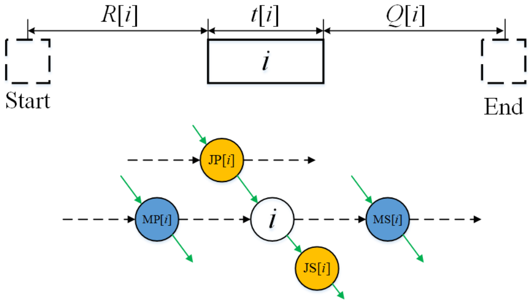


Figure 13: The illustration of R and Q

Estimation of insert move

Suppose u and v are the operations on the same machine, and u is on the left of v, after insert u into the rear of v, the makespan can be estimated as:

$makespan^{u,v} = \max\{R^{u,v}(w) + t[w] + Q^{u,v}(w)\}, w \in \{u, L_1, \ldots, L_k, v\}$

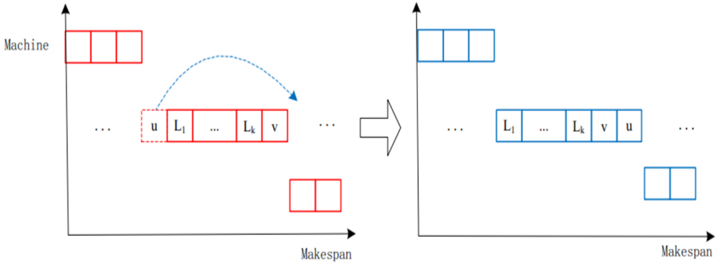


Figure 14: The illustration of insert move estimation

Estimation of R in backward insert move

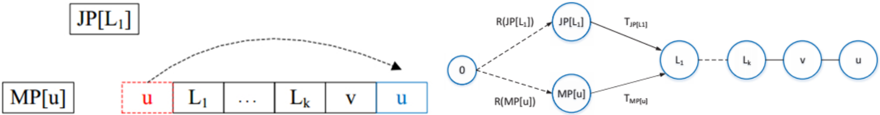


Figure 15: The illustration of backward insert move

For operation L_1,

$$R^{u,v}[L_1] = \begin{cases} R[JP[L_1]] + t[JP[L_1]], & \text{if } u \text{ is the first operation on the machine} \\ \max\{R[JP[L_1]] + t[JP[L_1]], R[MP[L_1]] + t[MP[L_1]]\}, & \text{otherwise} \end{cases} \quad (1)$$

For the other operations $w \in \{L_2, \dots, L_k, v\}$,
$R^{u,v}[w] = \max\{R[JP[w]] + t[JP[w]], R^{u,v}[MP[w]] + t[MP[w]]\}$;
For operation u,
$R^{u,v}[u] = \max\{R[JP[u]] + t[JP[u]], R^{u,v}[MP[v]] + t[MP[v]]\}$.

Estimation of Q in backward insert move

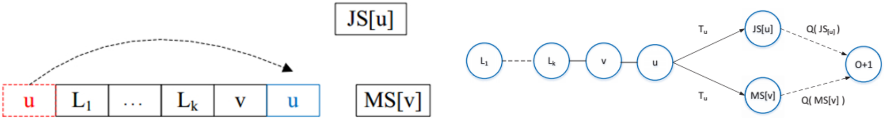


Figure 16: The illustration of backward insert move

For operation u,

$$Q^{u,v}[u] = \begin{cases} Q[JS[u]] + t[JS[u]], & \text{if } v \text{ is the first operation on the machine} \\ \max\{Q[JS[u]] + t[JS[u]], Q(MS[u]) + t[MS[u]]\}, & \text{otherwise} \end{cases} \quad (2)$$

For operation v,

$Q^{u,v}[v] = \max\{Q[JS[v]] + t[JS[v]], Q^{u,v}[u] + t[u]\}$

For the other operations $w \in \{L_1, \ldots, L_k\}$,

$Q^{u,v}[w] = \max\{Q[JS[w]] + t[JS[w]], Q^{u,v}(MS[w]) + t[MS[w]\};$

Estimation of R in forward insert move

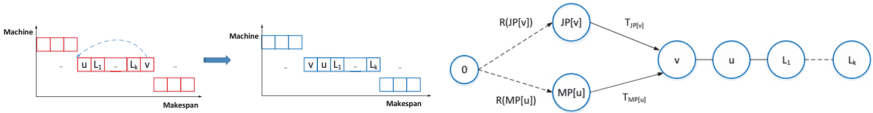


Figure 17: The illustration of forward insert move

For operation v,

$$R^{u,v}[v] = \begin{cases} R[JP[v]] + t[JP[v]], & \text{if } u \text{ is the first operation on the machine} \\ \max\{R[JP[v]] + t[JP[v]], R[MP[u]] + t[MP[u]]\}, & \text{otherwise} \end{cases} \quad (3)$$

For operation u,
$R^{u,v}[u] = \max\{R[JP[u]] + t[JP[u]], R^{u,v}[v] + t[v]\};$
For the other operations $w \in \{L_1, \ldots, L_k\}$,
$R^{u,v}[w] = \max\{R[JP[w]] + t[JP[w]], R^{u,v}[MP[w]] + t[MP[w]]\}.$

Estimation of Q in forward insert move

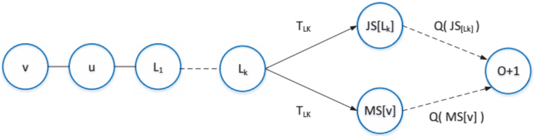


Figure 18: The illustration of forward insert move

For operation L_k,

$$Q^{u,v}[L_k] = \begin{cases} Q[JS[L_k]] + t[JS[L_k]], \text{if } v \text{ is the first operation on the machine} \\ \max\{Q[JS[L_k]] + t[JS[L_k]], Q[MS[v]] + t[MS[v]]\}, \quad \text{otherwise} \end{cases} \tag{4}$$

For the other operations $w \in \{u, L_2, \ldots, L_{k-1}\}$,
$Q^{u,v}[w] = \max\{Q[JS[w]] + t[JS[w]], Q^{u,v}[MS[w]] + t[MS[w]]\};$
For operation v,
$Q^{u,v}[v] = \max\{Q[JS[v]] + t[JS[v]], Q^{u,v}[u] + t[u]\}$

Makespan estimation in changing the machine assignment

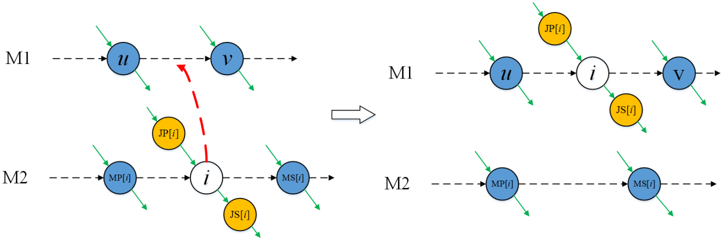


Figure 19: The illustration of changing machine assignment

Suppose i is the operation on machine M_i, u and v are the consecutive operations on machine M_u, move i out of M_i and insert it between u and v of M_u. Therefore, for operation i,

$R^{i,u,v}[i] = \max\{R[JP[i]] + t[JP[i]], R[u] + t[u]\};$

$Q^{i,u,v}[i] = \max\{Q[JS[i]] + t[JS[i]], Q[MS[v]] + t[MS[v]]\};$

$$makespan^{i,u,v} = \max\{R^{i,u,v}(i) + t[i] + Q^{i,u,v}(i)\}$$

An example of unfeasible move

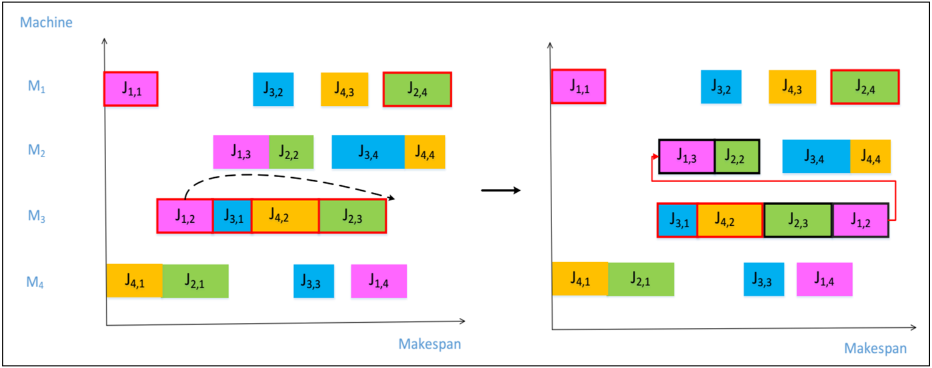


Figure 20: The illustration of unfeasible move

A cycle is encountered: J2,3 → J1,2 → J1,3 → J2,2 → J2,3

Theorem of feasible move

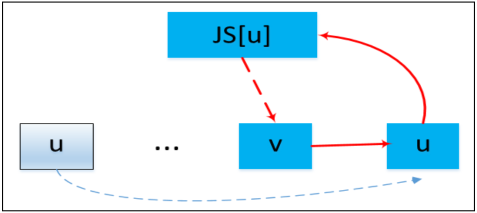


Figure 21: The illustration of feasible move

Theorem 1: *Suppose u and v are the critical operations on the same machine of a feasible solution, and u is on the left of v. If $Q[v] \geqslant Q[JS[u]]$, therefore, insert u after v results in a feasible solution.*

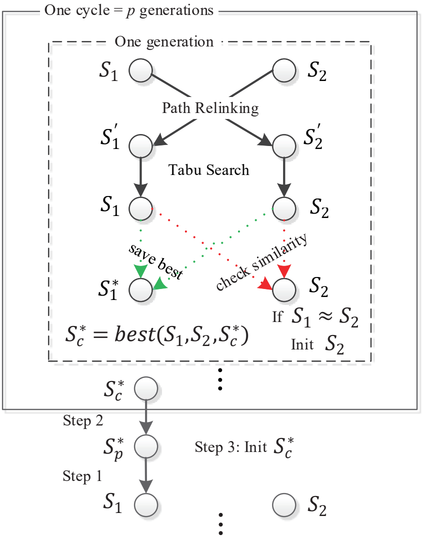


Figure 22: Diagram of MAE

MAE for FJSP

Algorithm 1 MAE, a two-individual based evolutionary algorithm for FJSP

Input: Problem instance
Output: The best solution S^* found
$gen \leftarrow 0$, $S_1, S_2, S_c^*, S_p^*, S^* \leftarrow$ Init()
while stopping condition is not reached **do**
  $S_1^{'} \leftarrow$ PR(S_1, S_2), $S_2^{'} \leftarrow$ PR(S_2, S_1)
  $S_1 \leftarrow$ TS$(S_1^{'})$, $S_2 \leftarrow$ TS$(S_2^{'})$
  $S_c^* \leftarrow$ save_best(S_1, S_2, S_c^*)
  $S^* \leftarrow$ save_best(S_c^*, S^*)
  if gen is equal to an integer parameter p **then**
    $S_1 \leftarrow S_p^*$, $S_p^* \leftarrow S_c^*$, $S_c^* \leftarrow$ Init(), $gen \leftarrow 0$
  end if
  if $S_1 \approx S_2$ **then**
    $S_2 \leftarrow$ Init()
  end if
  $gen \leftarrow gen + 1$
end while
return S^*

Distance definition

- Distance on the same machine: $d_o(S_1, S_2) = |P_o^{S_1} - P_o^{S_2}|$

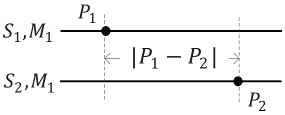


Figure 23: The illustration of the distance of the operation on the same machine.

Distance definition

- Distance on difference machines:
$d_o(S_1,S_2) = \min\{P_o^{S_1} + P_o^{S_2}, (L_{M_o^{S_1}}^{S_1} - P_o^{S_1}) + (L_{M_o^{S_2}}^{S_2} - P_o^{S_2})\}$

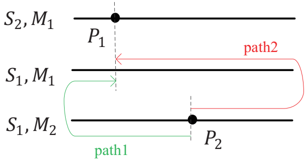


Figure 24: The illustration of the distance of the operation on the different machine.

Algorithm 2 A path relinking based recombination operator

Input: Initial solution S_i and guiding solution S_g
Output: A reference solution S_r
$S_c \leftarrow S_i, PathSet \leftarrow \emptyset, N \leftarrow \emptyset$
while $d(S_c, S_g) > \alpha \times d(S_i, S_g)$ **do**
  for each operation o in S_c **do**
    if $M_o^{Sc} \neq M_o^{Sg}$ **then**
      $N \leftarrow N \cup N_g^k(S_c, o)$
    else if $M_o^{Sc} = M_o^{Sg}$ and $P_o^{Sc} \neq P_o^{Sg}$ **then**
      $N \leftarrow N \cup N_g^{\pi}(S_c, o)$
    end if
  end for
  for each solution $S \in N$ **do**
    if $d(S, S_g) > d(S_c, S_g)$ **then** $N \leftarrow N \setminus \{S\}$
    else estimate makespan $obj(S)$ **end if**
  end for
  for each solution $S \in N$ **do**
    indexDis$(S) \leftarrow |\{T \in N | d(T, S_g) < d(S, S_g)\}|$
    indexObj$(S) \leftarrow |\{T \in N | obj(T) < obj(S)\}|$
  end for
  sort N in increasing order of indexDis(S)+indexObj(S), breaking ties randomly
  $k \leftarrow rand\{0, 1, \ldots, \min\{\gamma, |N| - 1\}\}$
  $S_c \leftarrow N(k)$; $N \leftarrow \emptyset$
  if $d(S_c, S_g) < \beta \times d(S_i, S_g)$ **then**
    $PathSet \leftarrow PahtSet \cup \{S_c\}$
  end if
end while
$S_r = \arg\min\{f(S), S \in PathSet\}$,
return S_r

Benchmark sets

Table 3: The descriptions of the benchmark sets

Set		n	m	*flex.*	*Size*	*#opt*
DPdata		[10, 20]	[5, 10]	[1.13, 5.02]	18	8
BCdata		[10, 15]	[11, 18]	[1.07, 1.30]	21	21
BRdata		[10, 20]	[5, 15]	[1.43, 4.10]	10	9
HUdata	*edata*	[6, 30]	[4, 10]	1.15	66	64
	rdata	[6, 30]	[4, 10]	2.00	66	54
	vdata	[6, 30]	[4, 10]	[2.50, 7.50]	66	65
	sdata	[6, 30]	[4, 10]	1.00	66	64
	Total				313	285

Comparison with the-state-of-the-art metaheuristics

- SSPR [González *et al.*, 2015] and GRASP-mELS [Kemmoé-Tchomté *et al.*, 2017] are the recent state-of-the-art metaheurstics for FJSP.
- Comparison are made in the same time limit.
- MAE improves the previous best results obtained by for 47 out of 178 benchmark instances while matching the best known results for others.
- MAE improves the previous best known results obtained by GRASP-mELS for 52 out of 178 benchmark instances and matches the best known results on all except 5 of the remaining instances.

Comparison with exact methods: summary

Table 4: Summary of MAE compared with CPO and Quintiq

Set		MAE(1 hour) vs CPO(8 hours)			MAE(1 hour) vs Quintiq		
		#better	#even	#worse	#better	#even	#worse
DPdata		13	5	0	3	2	10
BCdata		0	18	3	0	0	0
BRdata		2	8	0	0	2	0
HUdata	*edata*	2	60	4	0	20	0
	rdata	18	48	0	6	30	1
	vdata	10	53	3	1	22	0
	sdata	0	63	3	0	18	6
Total		45	255	13	10	94	17

Comparison with the exact method: CPO

Table 5: The improved results of MAE compared with CPO on 45 instances

Ins.	CPO		MAE	Ins.	CPO		MAE
	LB	UB	UB		LB	UB	UB
Mk05	168	173	172	rdata-la23	816	832	831
Mk10	183	195	193	rdata-la24	775	805	795
02a	2228	2234	2228	rdata-la25	752	787	779
05a	2189	2213	2203	rdata-la26	1056	1066	1057
06a	2162	2191	2181	rdata-la27	1085	1099	1086
07a	2206	2277	2254	rdata-la28	1075	1079	1076
08a	2061	2066	2062	rdata-la29	993	1001	994
10a	2197	2263	2245	rdata-la30	1068	1089	1071
11a	2017	2067	2045	rdata-la31	1520	1522	1520
12a	1969	2013	2008	rdata-la32	1657	1658	1657
13a	2161	2258	2236	rdata-la33	1497	1498	1497
14a	2161	2163	2162	rdata-la34	1535	1536	1535
16a	2148	2240	2232	vdata-car1	5005	5006	5005
17a	2088	2140	2121	vdata-car3	5597	5599	5597
18a	2057	2125	2103	vdata-car5	4909	4912	4910
edata-abz7	564	620	610	vdata-la22	733	734	733
edata-abz8	586	639	636	vdata-la25	751	753	752
rdata-abz7	492	535	522	vdata-la29	993	994	993
rdata-abz8	506	558	535	vdata-la30	1068	1069	1068
rdata-abz9	497	553	536	vdata-la32	1657	1658	1657
rdata-car3	5597	5623	5622	vdata-la33	1497	1498	1497
rdata-la21	808	838	825	vdata-la35	1549	1550	1549
rdata-la22	741	757	755				

New world records obtained by MAE

Table 6: New world records obtained by MAE

Ins.	Previous world record						MAE
	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date	UB
05a	2192	[Q]	Jan. 2014	2204	[Q]	Nov. 2015	2203
07a	2216	[CPO]	Dec. 2014	2264	[Q]	Nov. 2015	2254
13a	2197	[CPO]	Dec. 2014	2239	[Q]	Jan. 2016	2236
rdata-abz7	493	[Q]	Mar. 2013	524	[Q]	Jan. 2016	522
rdata-abz8	507	[Q]	Mar. 2013	536	[Q]	Jan. 2016	535
rdata-la22	741	[CPO]	Dec. 2014	755	[CPO]	Nov. 2013	753
rdata-la23	816	[Q]	Feb. 2000	832	[CPO]	Mar. 2013	831
rdata-la24	775	[Q]	Feb. 2000	796	[Q]	Nov. 2015	795
rdata-la25	768	[CPO]	Dec. 2014	783	[Q]	Jan. 2016	779
vdata-car5	4909	[Q]	Mar. 2013	4911	[Q]	Nov. 2015	4910

Comparison between MAE and the trajectory method

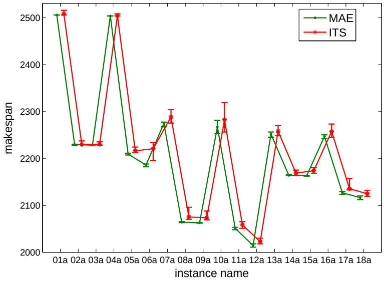


Figure 25: Comparison between MAE and ITS on *DPdata*.

The impact of the parameters: p

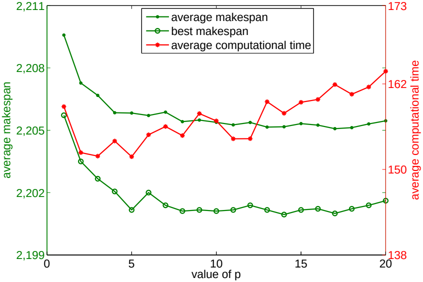


Figure 26: The average makespan and computational time corresponding to different values of parameter p on *DPdata*.

The impact of the parameters: α, β, γ

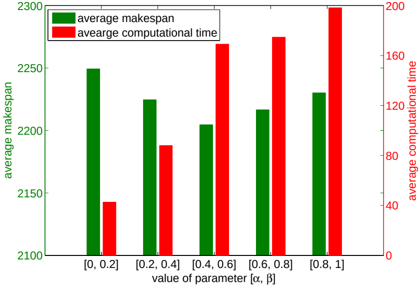


Figure 27: The average makespan and computational time corresponding to different values of parameter $[\alpha, \beta]$ on *DPdata*.

The impact of the parameters: γ

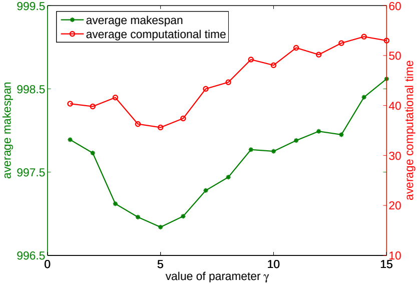
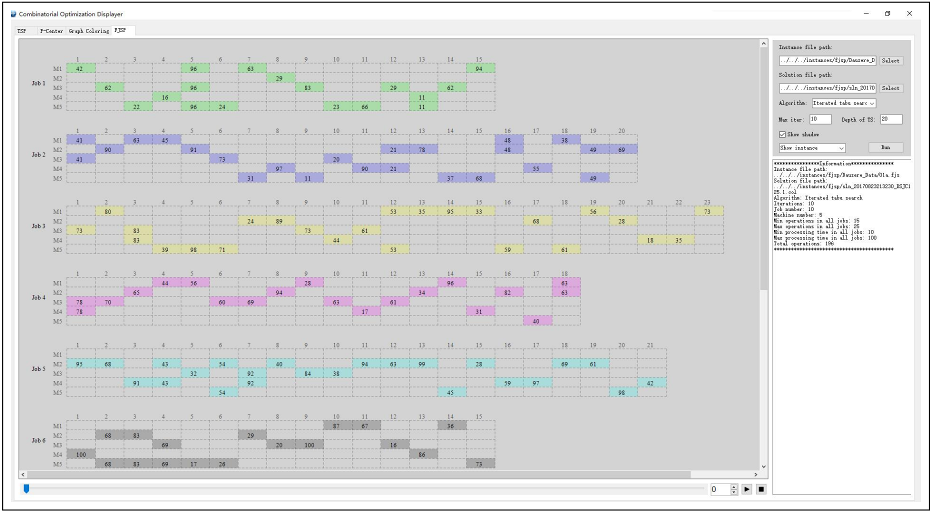
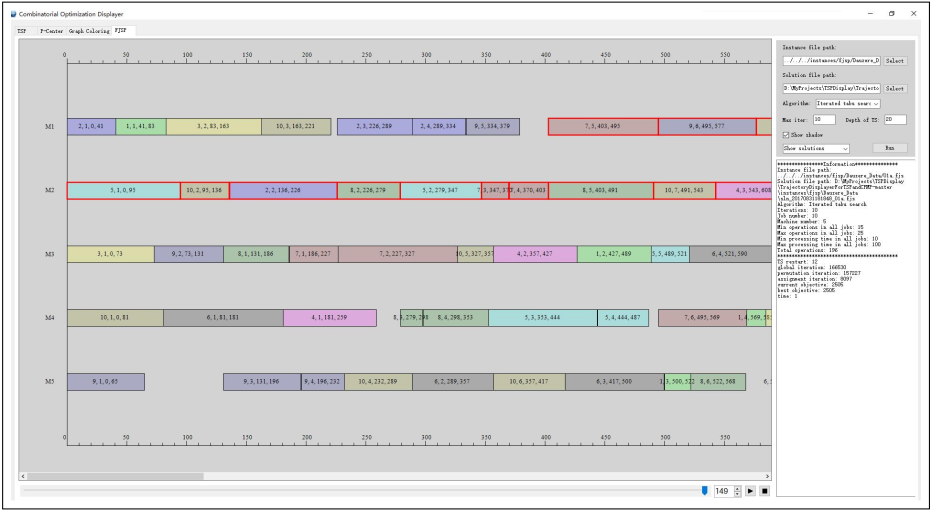


Figure 28: The average makespan and computational time corresponding to different values of parameter γ on *BCdata*.

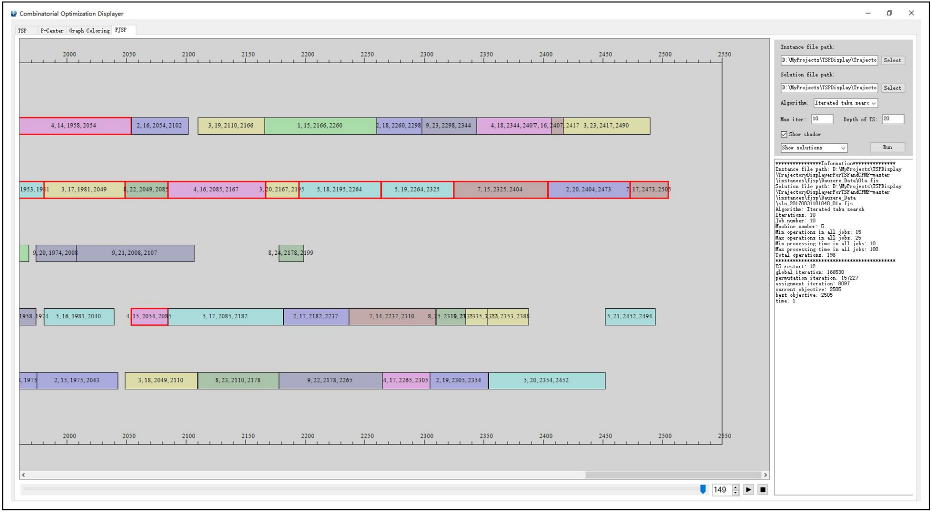
FJSP instance display 1



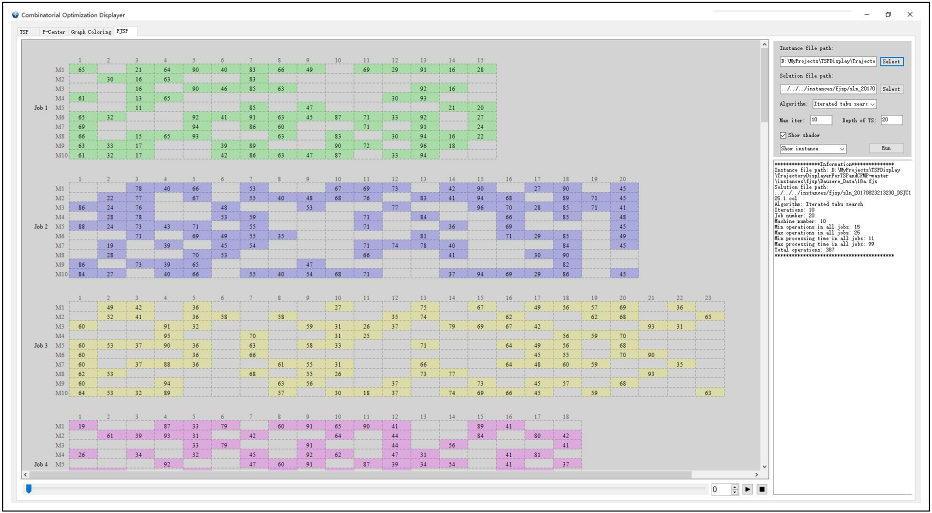
FJSP solution display 1



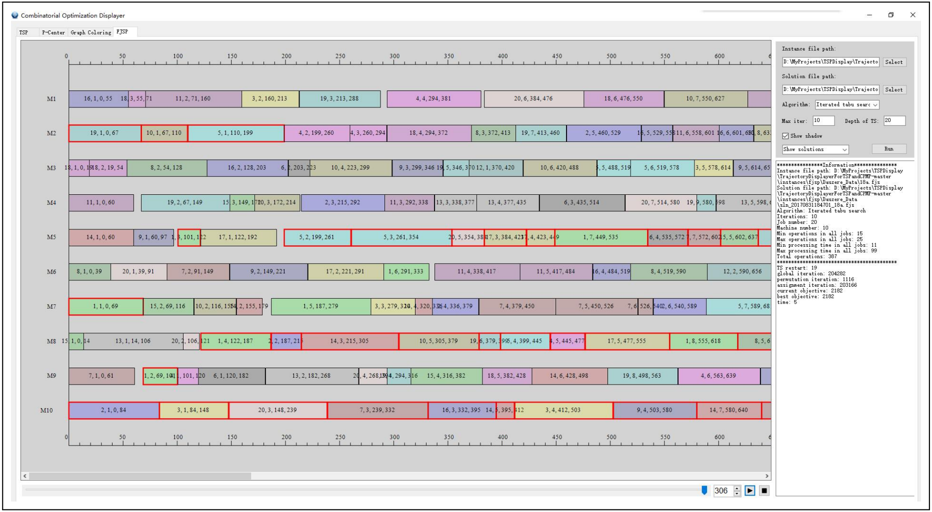
FJSP solution display 2



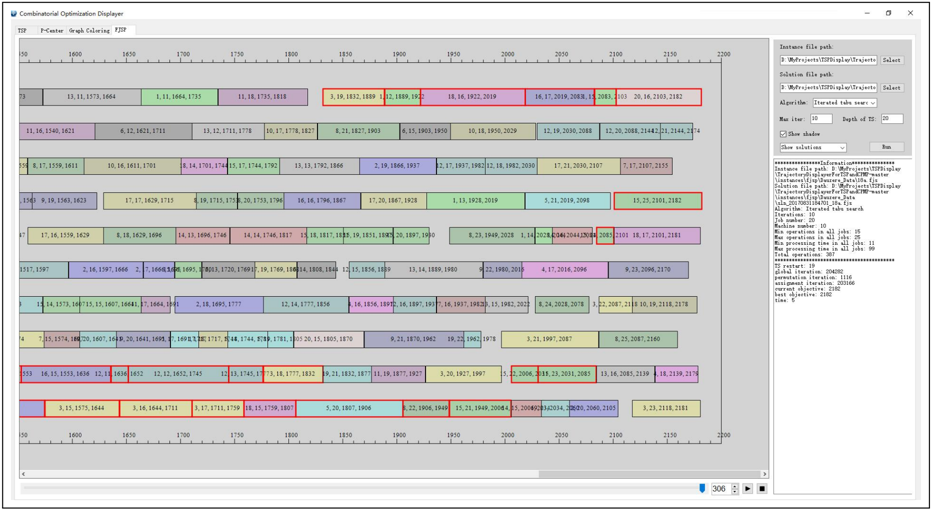
FJSP instance display 2



FJSP solution display 2



FJSP solution display 2



Complete solutions 1

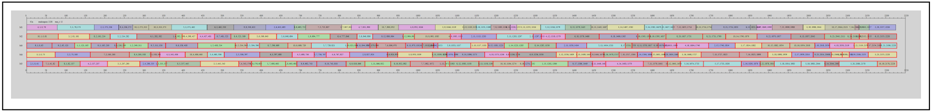


Figure 29: An optimal solution for instance $02a$

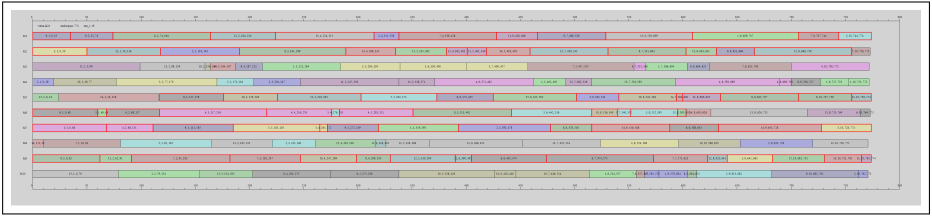


Figure 30: An optimal solution for instance $vdata - la24$

Complete solutions 2

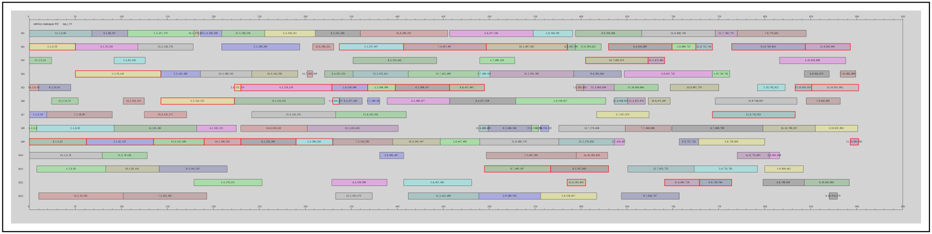


Figure 31: An optimal solution for instance $setb4xyz$

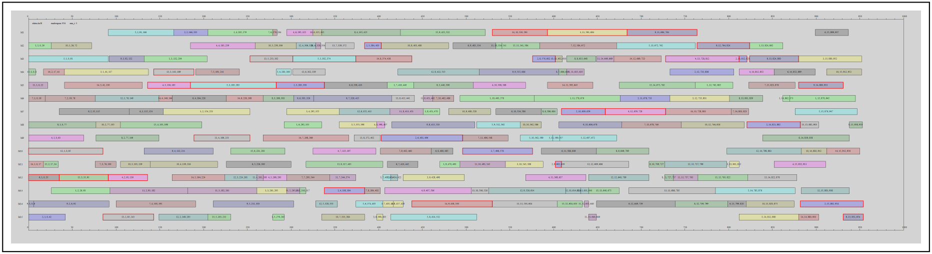
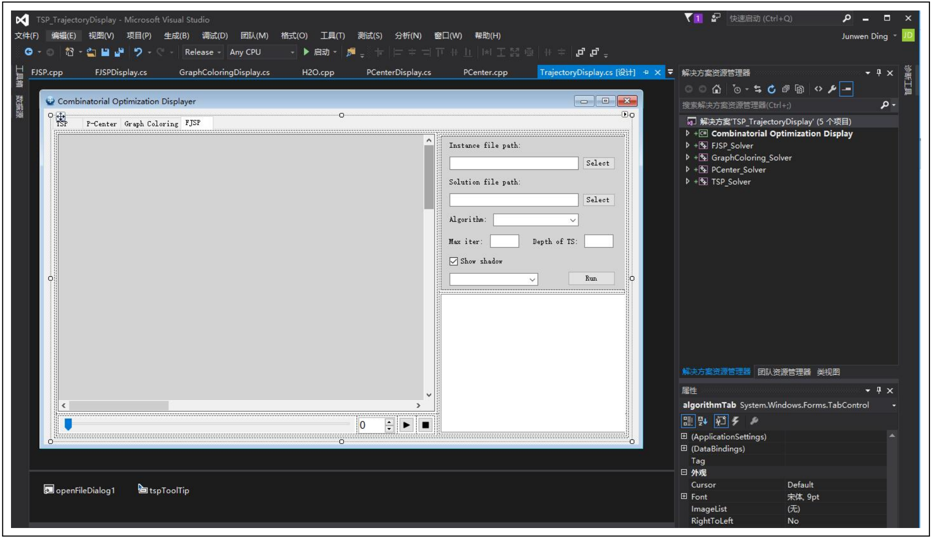


Figure 32: An optimal solution for instance $rdata-la38$

FJSP algorithm 1



FJSP algorithm 2

FJSP.cpp | FJSPDisplay.cs | H2O.cpp | PCenterDisplay.cs | PCenter.cpp | TrajectoryDisplay.cs [设计] | TSP_Ant.cpp | GraphColoringDisplay.cs | TrajectoryDisplay.cs

FJSP_Solver | → Solver | apply_permutation_move(int sol_index, int mach_i, int u, int v, MOVE_

```
void Solver::try_backward_insert_move(int sol_index, int &makespan, int mach_i, int u, int v)   // insert oper_u behind oper_v
{
    machine[sol_index][mach_i][u + 1]->apx_r = MAX(machine[sol_index][mach_i][u + 1]->pre_job_oper->end_time, machine[sol_index][mach_i][u - 1]->end_time);
    for (int oper_i = u + 2; oper_i <= v; oper_i++)
        machine[sol_index][mach_i][oper_i]->apx_r = MAX(machine[sol_index][mach_i][oper_i]->pre_job_oper->end_time, machine[sol_index][mach_i][oper_i - 1]->apx_r + machine[
    machine[sol_index][mach_i][u]->apx_r = MAX(machine[sol_index][mach_i][u]->pre_job_oper->end_time, machine[sol_index][mach_i][v]->apx_r + machine[sol_index][mach_i][v]->
    machine[sol_index][mach_i][u]->apx_q = MAX(machine[sol_index][mach_i][u]->next_job_oper->q + machine[sol_index][mach_i][u]->next_job_oper->t, machine[sol_index][mach_i]
    machine[sol_index][mach_i][v]->apx_q = MAX(machine[sol_index][mach_i][v]->next_job_oper->q + machine[sol_index][mach_i][v]->next_job_oper->t, machine[sol_index][mach_i]
    for (int oper_i = v - 1; oper_i > u; oper_i--)
        machine[sol_index][mach_i][oper_i]->apx_q = MAX(machine[sol_index][mach_i][oper_i]->next_job_oper->q + machine[sol_index][mach_i][oper_i]->next_job_oper->t, machine
    makespan = 0;
    for (int oper_i = u; oper_i <= v; oper_i++)
        if (makespan < machine[sol_index][mach_i][oper_i]->apx_r + machine[sol_index][mach_i][oper_i]->apx_q + machine[sol_index][mach_i][oper_i]->t)
            makespan = machine[sol_index][mach_i][oper_i]->apx_r + machine[sol_index][mach_i][oper_i]->apx_q + machine[sol_index][mach_i][oper_i]->t;
}
void Solver::try_forward_insert_move(int sol_index, int &makespan, int mach_i, int u, int v)    // insert oper_v before oper_u
{
    machine[sol_index][mach_i][v]->apx_r = MAX(machine[sol_index][mach_i][v]->pre_job_oper->end_time, machine[sol_index][mach_i][u - 1]->end_time);
    machine[sol_index][mach_i][u]->apx_r = MAX(machine[sol_index][mach_i][u]->pre_job_oper->end_time, machine[sol_index][mach_i][v]->apx_r + machine[sol_index][mach_i][v]->
    for (int oper_i = u + 1; oper_i < v; oper_i++)
        machine[sol_index][mach_i][oper_i]->apx_r = MAX(machine[sol_index][mach_i][oper_i]->pre_job_oper->end_time, machine[sol_index][mach_i][oper_i - 1]->apx_r + machine[
    machine[sol_index][mach_i][v - 1]->apx_q = MAX(machine[sol_index][mach_i][v - 1]->next_job_oper->q + machine[sol_index][mach_i][v - 1]->next_job_oper->t, machine[sol_in
    for (int oper_i = v - 2; oper_i >= u; oper_i--)
        machine[sol_index][mach_i][oper_i]->apx_q = MAX(machine[sol_index][mach_i][oper_i]->next_job_oper->q + machine[sol_index][mach_i][oper_i]->next_job_oper->t, machine
    machine[sol_index][mach_i][v]->apx_q = MAX(machine[sol_index][mach_i][v]->next_job_oper->q + machine[sol_index][mach_i][v]->next_job_oper->t, machine[sol_index][mach_i]
    makespan = 0;
    for (int oper_i = u; oper_i <= v; oper_i++)
        if (makespan < machine[sol_index][mach_i][oper_i]->apx_r + machine[sol_index][mach_i][oper_i]->apx_q + machine[sol_index][mach_i][oper_i]->t)
            makespan = machine[sol_index][mach_i][oper_i]->apx_r + machine[sol_index][mach_i][oper_i]->apx_q + machine[sol_index][mach_i][oper_i]->t;
}
// u is less than v, move u after v, or move v before u
void Solver::apply_permutation_move(int sol_index, int mach_i, int u, int v, MOVE_TYPE move_type)
{
    if (move_type == BACKWARD_INSERT)
    {
        Solution::Operation *oper_u = machine[sol_index][mach_i][u];
        for (int i = u; i < v; i++)
        {
            machine[sol_index][mach_i][i] = machine[sol_index][mach_i][i + 1];
```

100 %

References

Miguel Ángel. González, Camino Rodríguez Vela, and Ramiro Varela. Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, 245(1):35–45, 2015.

Sylverin Kemmoé-Tchomté, Damien Lamy, and Nikolay Tchernev. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. *Engineering Applications of Artificial Intelligence*, 62:80–95, 2017.

Chao Yong Zhang, Pei Gen Li, Zai Lin Guan, and Yun Qing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.