

On Job-Insertion for the Blocking-Job-Shop and its Application to the SBB Challenge

Master thesis of

Fabio Degiacomi

Supervised by

Dr. Reinhart Bürgy & Prof. Bernhard Ries

Decision Support & Operations Research Group,
University of Fribourg

September 2020

References

- ▶ The thesis on which this presentation is based is found on:
<https://github.com/MrPascalCase/SbbChallenge>
- ▶ All references are found
→ Thesis.pdf → References
- ▶ Complete definitions and the introduction of all notation is also found in the thesis.

Job-Shop Problem (i)

The ‘input’ data:

- ▶ a set of *jobs* $\mathcal{J} = \{J_1, \dots, J_n\}$,
- ▶ each job is a sequence of *operations*, i.e.
 $J = (o_{J1}, \dots, o_{Jn_J})$,
- ▶ each operation o has a processing time p_o ,
- ▶ a set of *machines* $\mathcal{M} = \{M_1, \dots, M_m\}$,
- ▶ each operation o has a machine associated $M(o)$.

The problem, define a time t_o for every operation o :

$$\min_t \max_{J \in \mathcal{J}} \{t_{Jn_J} + p_{Jn_J}\}$$

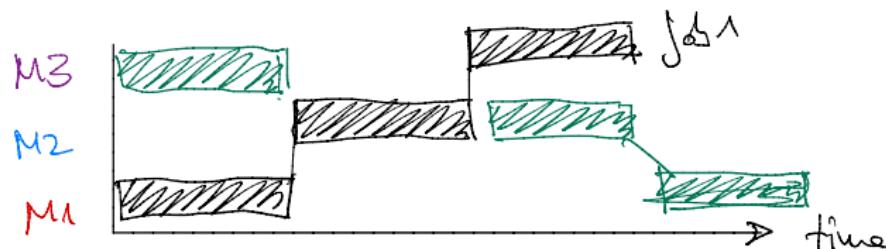
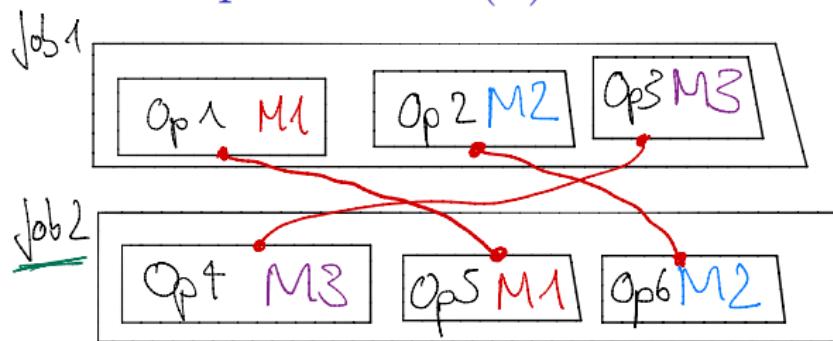
$$0 \leq t_o \quad \forall o \in \mathcal{O}$$

$$t_{Jk} + p_{Jk} \leq t_{Jl} \quad \forall J \in \mathcal{J} \ \forall k \ \forall l \mid 1 \leq k < l \leq n_J$$

$$t_{o_1} + p_{o_1} \leq t_{o_2} \vee t_{o_2} + p_{o_2} \leq t_{o_1}$$

$$\forall \{(o_1, o_2) \mid o_1 \in \mathcal{O}, o_2 \in \mathcal{O}, o_1 \neq o_2, M(o_1) = M(o_2)\}$$

Job-Shop Problem (ii)



Blocking-Job-Shop Problem

The ‘input’ data:

- ▶ a set of *jobs* $\mathcal{J} = \{J_1, \dots, J_n\}$,
- ▶ each job is a sequence of *operations*, i.e.
 $J = (o_{J1}, \dots, o_{Jn_J})$,
- ▶ each operation o has a processing time p_o ,
- ▶ a set of *machines* $\mathcal{M} = \{M_1, \dots, M_m\}$,
- ▶ each operation o has a machine associated $M(o)$.

The problem, define a time t_o for every operation o :

$$\min_t \max_{J \in \mathcal{J}} \{t_{Jn_J} + p_{Jn_J}\}$$

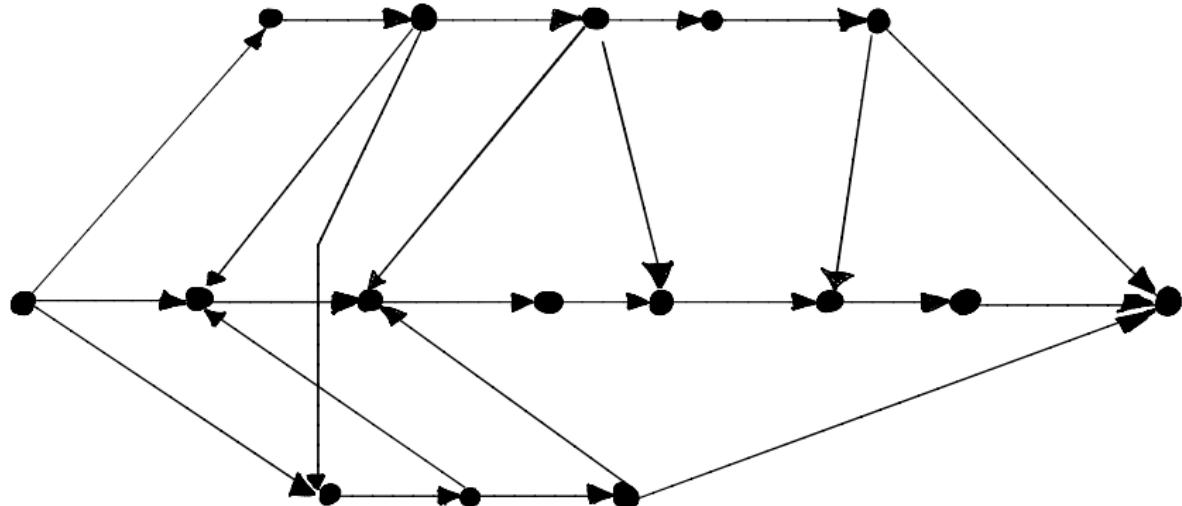
$$0 \leq t_o \quad \forall o \in \mathcal{O}$$

$$t_{Jk} + p_{Jk} \leq t_{Jl} \quad \forall J \in \mathcal{J} \ \forall k \ \forall l \mid 1 \leq k < l \leq n_J$$

$$t_{succ(o_1)} \leq t_{o_2} \vee t_{succ(o_2)} \leq t_{o_1}$$

$$\forall \{(o_1, o_2) \mid o_1 \in \mathcal{O}, o_2 \in \mathcal{O}, o_1 \neq o_2, M(o_1) = M(o_2)\}$$

Precedence constraint graph



Disjunctive graph (i)

Definition. A tuple $(V, A, E, \mathcal{E}, l)$ defines a disjunctive graph, if

- (i) $(V, A \cup E)$ is a directed graph.
- (ii) (V, A) is an acyclic directed graph.
- (iii) \mathcal{E} is a set of unordered pairs of arcs; E exactly contains all arcs of \mathcal{E} , i.e.:

$$\forall (e, \bar{e}) \in \mathcal{E} \ (e \in E \wedge \bar{e} \in E).$$

Furthermore, adding both arcs of a pair to (V, A) yields a cyclic graph. i.e.:

$$\forall (e, \bar{e}) \in \mathcal{E} \ ((V, A \cup \{e, \bar{e}\}) \text{ is cyclic.})$$

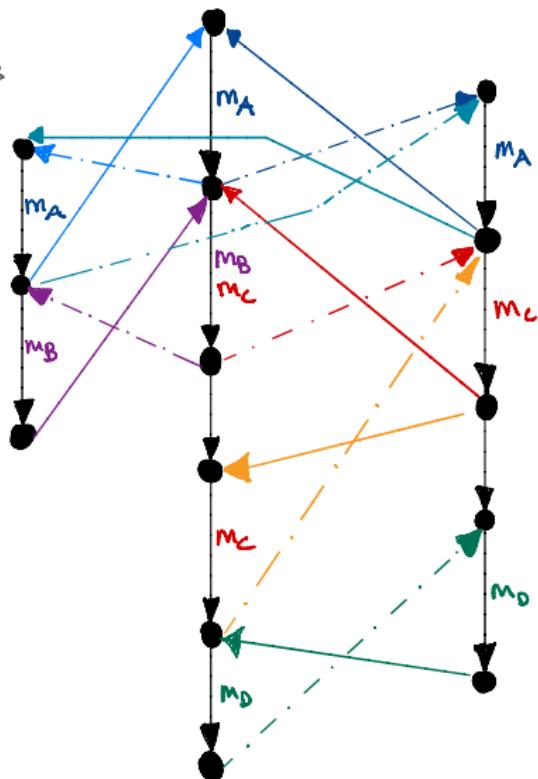
For $(e, \bar{e}) \in \mathcal{E}$, we call \bar{e} the mate of e and vice versa.

- (iv) $l : (A \cup E) \rightarrow \mathbb{R}_{>0}$, defines the length of an arc.

$a \in A$ is called conjunctive, $e \in E$ is called disjunctive.

Disjunctive graph (ii)

Vertices
 $(V, A, E, \mathcal{E}, l)$
conjunctive arcs
disjunctive arcs
pairs of disjunctive arcs



Selection (i)

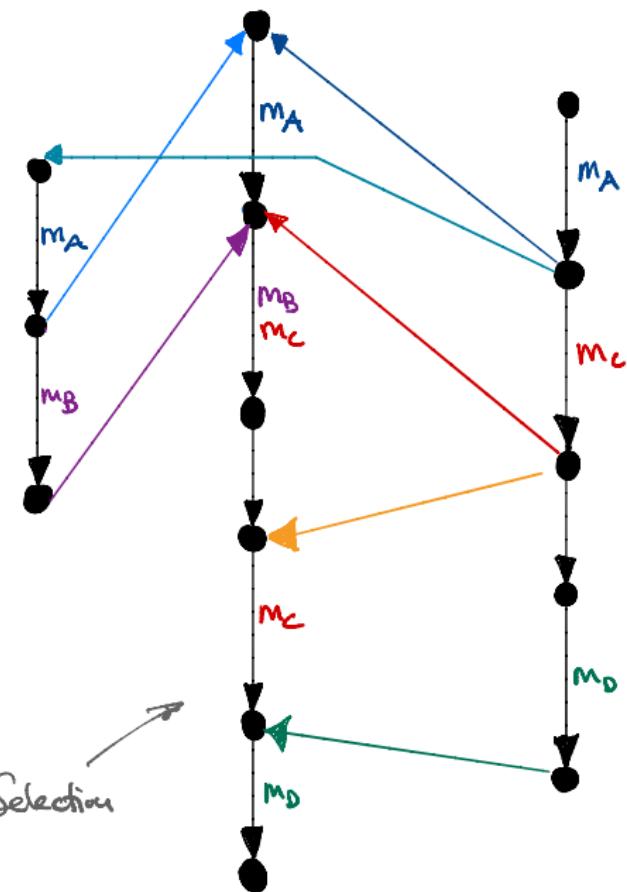
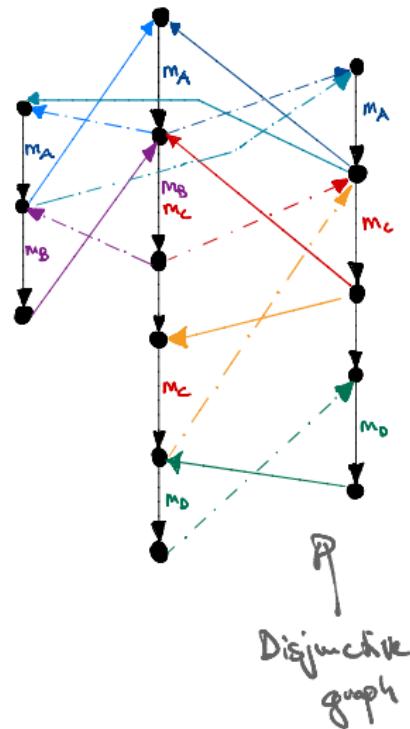
Definition. S is a selection in a disjunctive graph $G = (V, A, E, \mathcal{E}, l)$ if

- ▶ $S \subset E$
- ▶ S contains at most one element of every pair of \mathcal{E} .

We call S complete (else partial) if S contains one element of every pair of \mathcal{E} .

S is called feasible if $(V, A \cup S)$ is acyclic.

Selection (ii)



Selection (iii)

Goal:

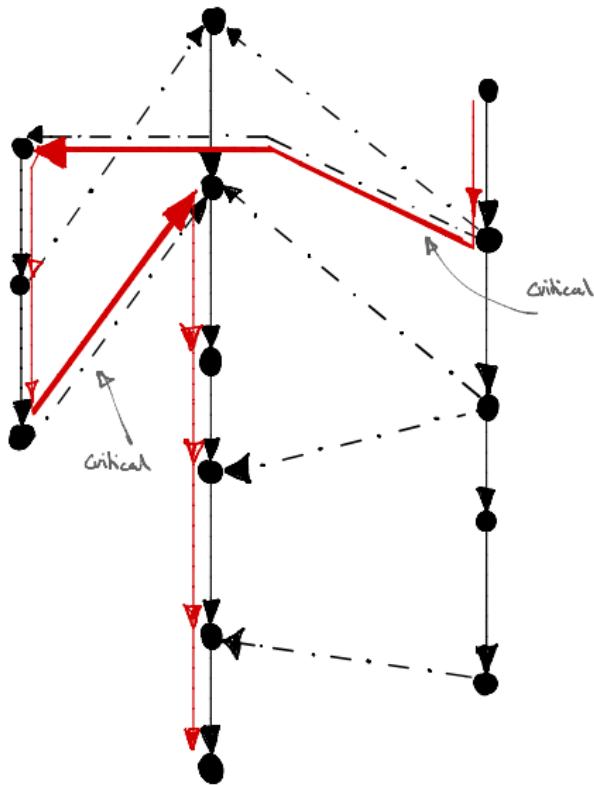
Given a complete, feasible selection, create ‘neighbour’ selections, such that we can apply a meta search heuristic (such as a taboo-search).

- ▶ In the (classical-)job-shop exchanging any e with \bar{e} yields such a neighbour.
- ▶ Not in the blocking-job-shop. How can we ‘repair’ a selection after such a change?
- ▶ Context:

$$\max_x \{f(x) \mid x \in \{0, 1\}^n\}, \\ N_1(x) = \{y \mid \|x - y\|_1 = 1\}.$$

Critical arcs (i)

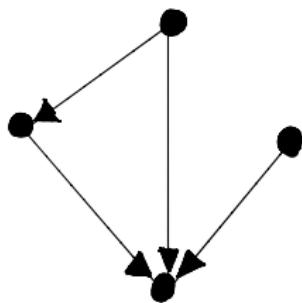
Definition. A disjunctive arc along a longest path in the graph is called critical.



Selection $S \rightarrow$ entry times t

- ▶ A selection corresponds to a set of feasible schedules. One of which, the *earliest-starting-date-schedule*, is among the best (cf. regular objective).
- ▶ For all $v \in V$, in topological order, set:

$$t_v = \max\{ t_u + l(u, v) \mid (u, v) \in (A \cup S) \}$$



- ▶ Disregarding other schedules, we have a one-to-one relation.

Job-Insertion Graph (i)

Definition. Given a blocking-job-shop problem with jobs \mathcal{J} , a specific job $J \in \mathcal{J}$ and a complete, feasible selection S to the problem $\mathcal{J} \setminus J$. For S i.e.:

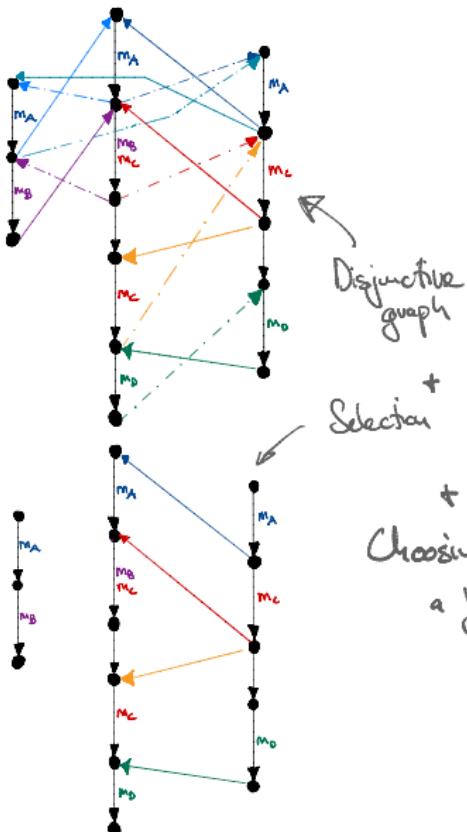
- ▶ $\forall(e, \bar{e}) \in \mathcal{E}$ not adjacent to J , $e \in S \vee \bar{e} \in S$.
- ▶ $\forall(e, \bar{e}) \in \mathcal{E}$ adjacent to J , $e \notin S \wedge \bar{e} \notin S$.

Associated with the blocking-job-shop problem is the graph $G = (V, A, E, \mathcal{E}, l)$.

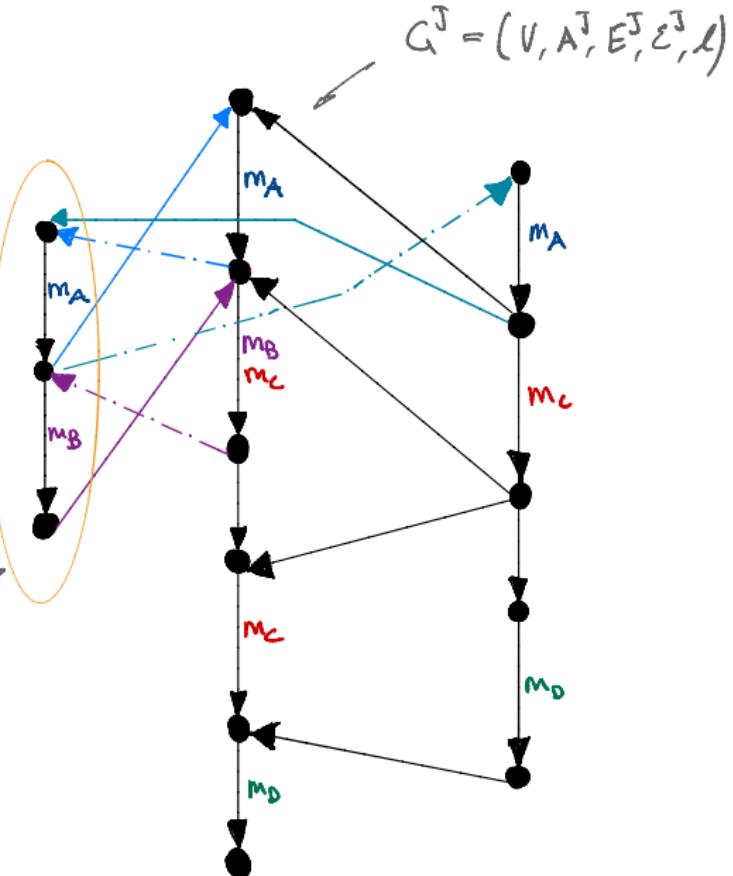
The job-insertion graph $G^J = (V, A^J, E^J, \mathcal{E}^J, l)$ is constructed as follows:

- ▶ $A^J := A \cup S$
- ▶ $\mathcal{E}^J := \mathcal{E}$ restricted to arcs adjacent to J .
- ▶ $E^J := E$ restricted to arcs adjacent to J .

Job-Insertion Graph (ii)

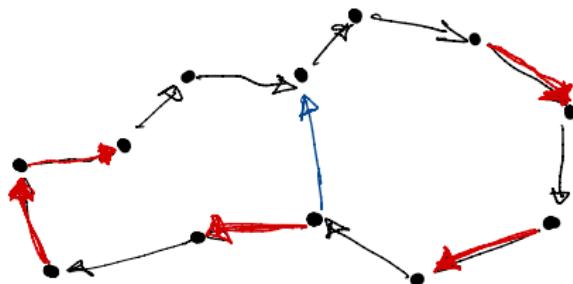


+
Selection
+
Choosing a job J



Short-Cycle Property (i)

Definition. A disjunctive graph $G = (V, A, E, \mathcal{E}, l)$ has the short-cycle property if for any cycle Z in $(V, A \cup E)$, there exists a cycle Z' in $(V, A \cup E)$ with $Z' \cap E \subseteq Z \cap E$ and $|Z' \cap E| = 2$.



Short-Cycle Property (ii)

Proposition. The job-insertion graph $G^J = (V, A^J, E^J, \mathcal{E}^J, l)$ has the short cycle property.

- ▶ **(n = 1)** Enter and leave J , $|Z \cap E| = 2$. ✓
- ▶ **(n ↠ n + 1)** Prove that: a cycle entering J $n + 1$ times has a short cycle.

Let Z be a cycle which enters J $n + 1$ times. We choose arbitrarily a vertex a where the cycle Z leaves J .

Let b be the first vertex after a where Z reenters J . We differentiate two cases:

(i) **a ⊲ b:**

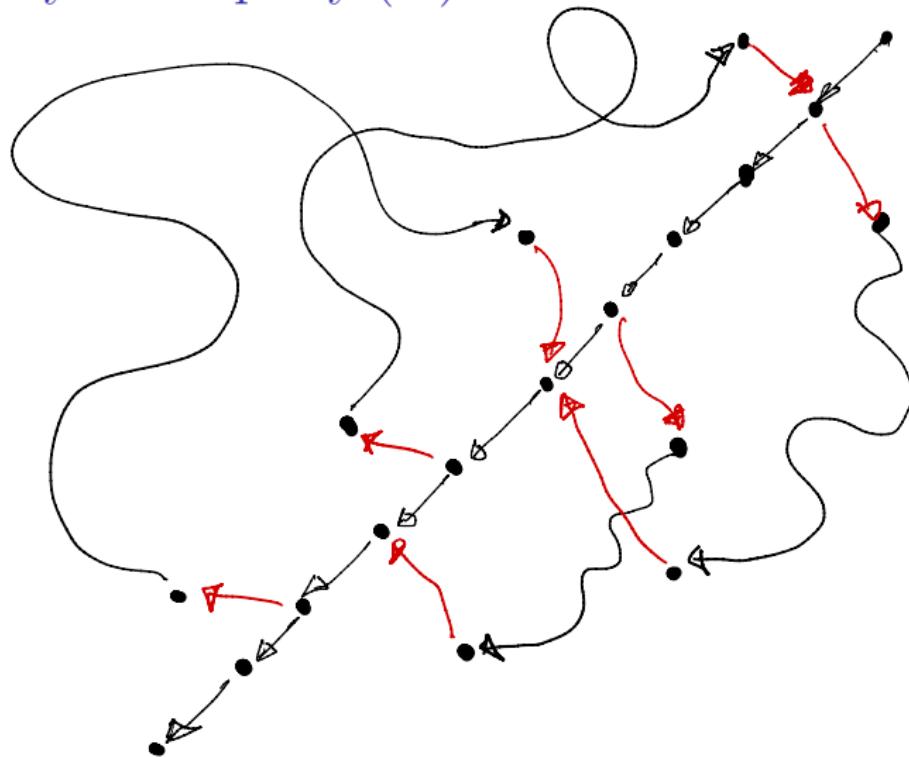
Define a cycle Z' , equal to Z , with the path $a \rightarrow b$ replaced by the path $a \rightarrow b$ within J . Then Z' enters J n times.

From the induction hypothesis it follows that a short cycle exists. ✓

(ii) **¬(a ⊲ b):**

$b \rightarrow a$ (within Z) followed by $a \rightarrow b$ (within J) is a valid short cycle. ✓

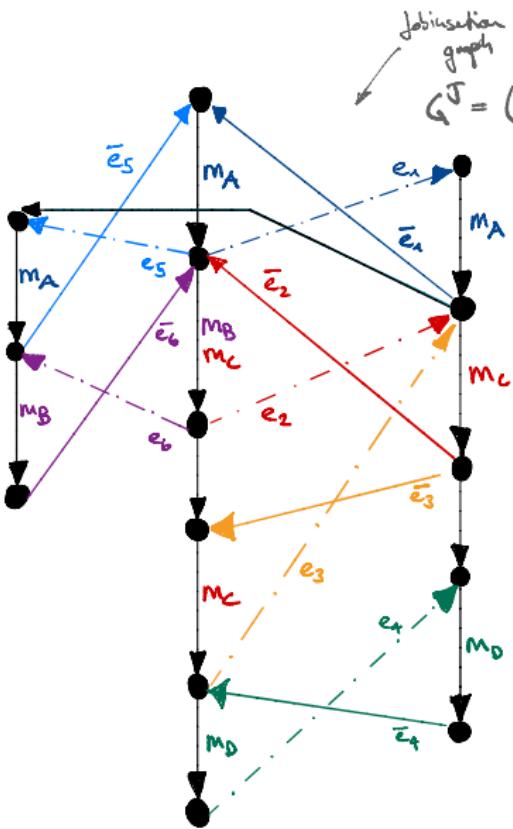
Short-Cycle Property (iii)



Conflict graph (i)

- ▶ We define a conflict graph for a job-insertion graph G^J :
 $H_{G^J} = (E^J, U)$.
- ▶ Vertices of H_{G^J} are elements to be selected $\in E^J$.
- ▶ Vertices conflict \implies connected. I.e. selecting all vertices connected by an arc u leads to a cyclic graph $(V, A \cup u)$.
- ▶ $u \in U \implies u$ is a partial infeasible selection.
- ▶ As we require a feasible solution, it suffices to consider minimally infeasible edges.
- ▶ The short cycle property guarantees that all edges connect *two* vertices.
- ▶ Hence, $U = \{(e, f) \mid (V, A^J \cup \{e, f\}) \text{ cyclic}\}$.
- ▶ (H_{G^J} is bipartite.)
- ▶ Stable sets of size $|E^J|/2$ in the conflict graph correspond to complete, feasible solutions.

Conflict graph (ii)

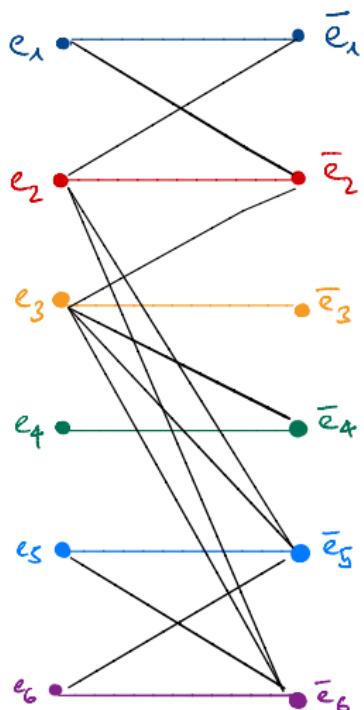


Job insertion
graph

$$G^J = (V, A^J, E^J, \Sigma^J, \ell)$$

Conflict graph

$$H_{G^J} = (E^J, U)$$



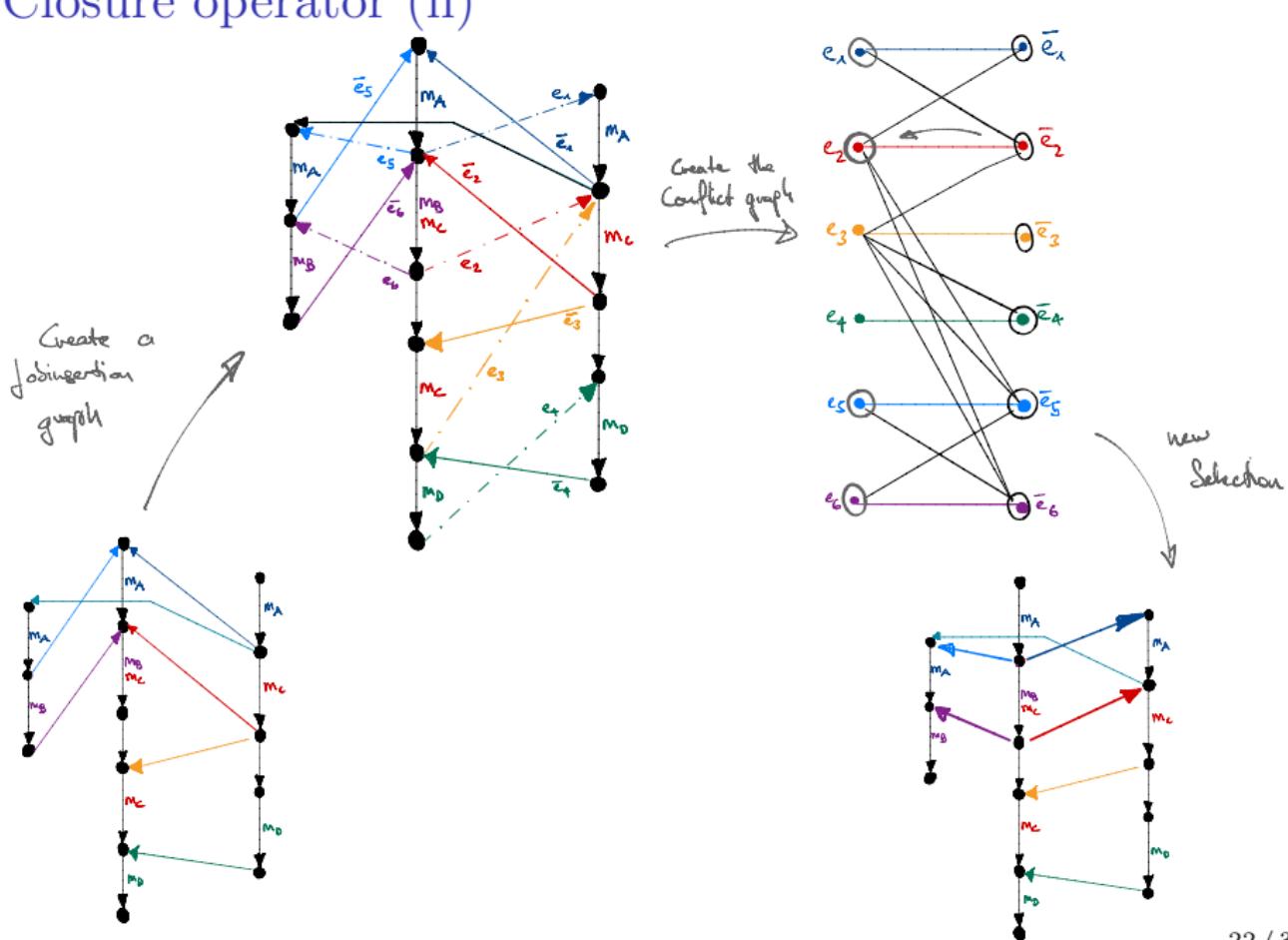
Closure operator (i)

Used to generate a complete, feasible selection S' from S which does not include $e \in S$.

Idea:

- ▶ Choose $J \in \mathcal{J}$ with $h(e) \in J$.
- ▶ Construct G^J .
- ▶ Construct H_{G^J} .
- ▶ Select \bar{e} in H_{G^J} .
- ▶ For all f such that (\bar{e}, f) in U , select \bar{f} . (*)
- ▶ Take the closure of (*).
- ▶ Complement this selection with S for all pairs in \mathcal{E}^J where we did not make a choice due to the closure.

Closure operator (ii)



Why do we need the conflict graph? (i)

- ▶ Query information like ‘enumerate the neighbours of e ’.
- ▶ In the disjunctive graph, this is equivalent to ‘find all disjunctive arcs $f \in E^J$ such that $(V, A^J \cup \{e, f\})$ is cyclic’.
- ▶ This corresponds to paths searches:
for all $\{f \in E^J \mid h(f) \preceq t(e)\}$ does a path $h(e) \rightarrow t(f)$ exist?

Why do we need the conflict graph? (ii)

‘Left’ Closure (i)

- ▶ Constructing G^J we picked J such that $h(e) \in J$.
- ▶ (We could do the same thing with J such that $t(e) \in J$
 \implies ‘right’ closure.)
- ▶ In case of the left-closure: Arcs such as e which *enter* J are replaced by arcs \bar{e} that *leave* J . Whenever we insert \bar{e} , we have to check if a path exists between $h(\bar{e}) \rightarrow t(\bar{e})$ ($t(\bar{e}) \in J$, $h(\bar{e}) \notin J$). Of a potential cycle, only the disjunctive arc reentering J is relevant. Hence we search paths from $h(\bar{e})$ into the vertex interval $[o_{J1}, \dots, t(\bar{e})]$. This is done efficiently with a forward path search.

‘Left’ Closure (ii)

- ▶ We remove an arc e which *enters* J while adding an arc \bar{e} that *leaves* J .
- ▶ A short cycle through \bar{e} must therefore contain an arc f which *enters* J .
- ▶ f must be replaced with \bar{f} , which *leaves* J .
- ▶ The same argument holds for \bar{f} , indeed for the complete process. More and more arcs leave J , while fewer and fewer arcs enter J . This has the effect that J moves backward in time through the time table, or, in a Gantt-chart, to the left. We call this version of the closure the left-closure.

Algorithm 1: Naive left-closure

Input : A graph G corresponding to the current selection

$S: G = (V, A \cup S)$, a disjunctive arc a to remove from the selection.

Output: A modified graph G , corresponding to a modified selection S , which is similar to the input S but does not contain the arc a .

Set<Arc> arcsToRemove = { a }

while $arcsToRemove.Count > 0$ **do**

 Arc $e = arcsToRemove.Pop()$

if $G.ArcExists(e)$ **then**

 Arc $\bar{e} = G.SwapInMate(e)$

foreach Arc $f \in$

$G.ForwardPathSearchInRange(h(\bar{e}), [o_{J1}, t(\bar{e})])$ **do**

 arcToRemove.Add(f)

end

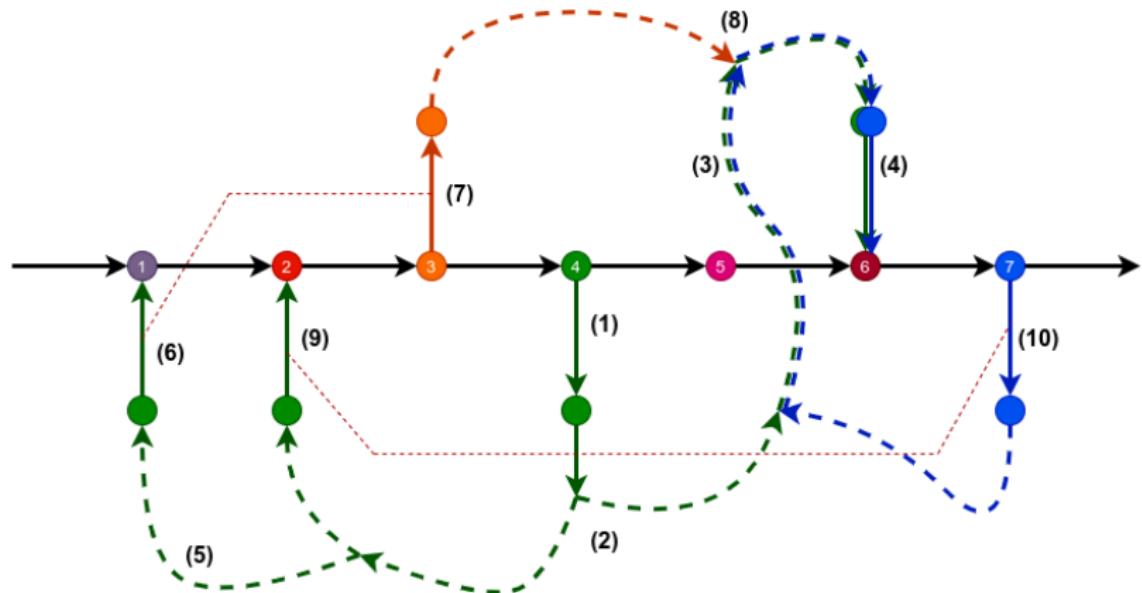
end

end

My contribution (i)

Observe that, the later within J the operation $t(\bar{e})$ occurs, the larger the target vertex interval of the path search. Indeed, $t(\bar{f}) \preceq t(\bar{e})$ implies $\{o_{J1}, \dots, t(\bar{f})\} \subseteq \{o_{J1}, \dots, t(\bar{e})\}$. Hence, for any vertex $v \in G$ such that a path exists from $h(\bar{f}) \rightarrow v$ and $h(\bar{e}) \rightarrow v$, the forward path search (v onward) to be completed for \bar{e} renders the path search for \bar{f} irrelevant.

My contribution (ii)



Algorithm 2: All at once left-closure

Input : A disjunctive graph G and a selection S :

$G = (V, A \cup S)$, a disjunctive arc a to remove from
the selection.

Output: A modified graph G , corresponding to a modified
selection S , which is similar to the input S but
does not contain the arc a .

[...]

Algorithm 3: All at once left-closure

[...]

Queue<Arc> Q = new BucketQueue()

let \bar{a} be s.t. $\bar{a} \in G.\text{SwapInMate}(a)$ **and** $t(\bar{a}) \in J$

Q.Add(\bar{a} , priority = any)

int[] P = new int[G.Count]

Initialize P: $P[o_{jk}] = \begin{cases} k, & \text{if } j = J \\ 0, & \text{otherwise} \end{cases}$

[...]

Algorithm 4: All at once left-closure

[...]

```
while  $Q.Count > 0$  do
    Arc  $a = Q.Pop()$ 
    if not  $G.ArcExists(a)$  then
        continue
    else if  $h(a) \in J$  and  $P[t(a)] \geq P[h(a)]$  then
        foreach Arc  $b \in G.SwapInMate(a)$  do
            if  $P[t(b)] > 0$  then
                |  $Q.Add(b, \text{priority} = P[t(b)])$ 
            end
        end
    else if  $P[t(a)] > P[h(a)]$  then
         $P[h(a)] = P[t(a)]$ 
        foreach Arc  $b \in G.OutgoingArc(h(a))$  do
            |  $Q.Add(b, \text{priority} = P[t(b)])$ 
        end
    end
end
```

Another improvement

- ▶ For a time t , we split $(V, A \cup S)$ into

$$A := \{v \mid t_v \leq t\} \text{ and } B := \{v \mid t_v > t\}.$$

- ▶ No vertex of A is reachable from any vertex in B .

Assuming that the left-closure computation affects no arcs adjacent to or in B , we can establish that A remains unreachable from B . If we then—during the forward path search of the left closure—encounter a vertex of B we can skip this branch of the path search.

- ▶ We choose $t = \max_{a \in \delta(J)} \{ t_{h(a)} \}$.

Algorithm 5: All at once left-closure with termination criterion

[...]

Initialize B: $B[i] = \begin{cases} true, & \text{if } t_i > \max_{b \in \delta(J)} \{ t_{h(b)} \} \\ false, & \text{otherwise} \end{cases}$

while $Q.Count > 0$ **do**

 Arc $a = Q.Pop()$

 [...]

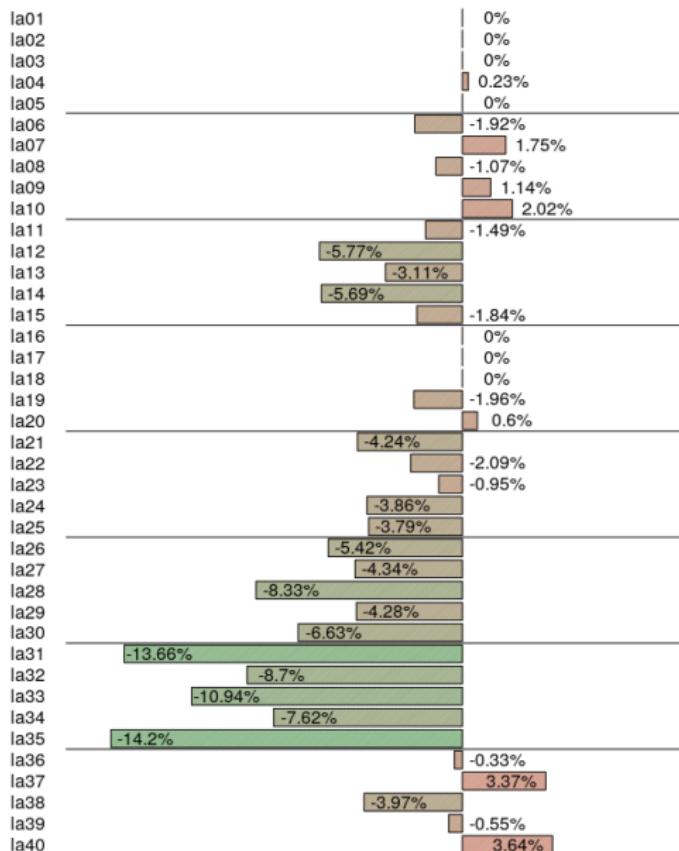
else if $B[h(a)]$ **then**

continue

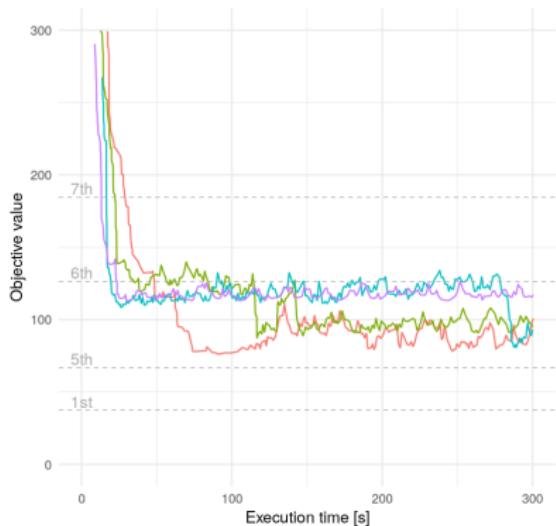
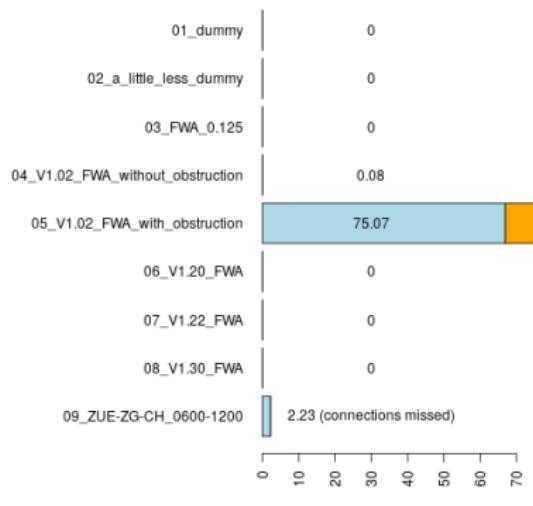
 [...]

end

Results (i)



Results (ii)



Conclusion

- ▶ ‘Academic datasets’: Compared to Pranzo & Pacciarelli, on larger instances clearly superior.
- ▶ SBB datasets: Fast, but worse than other submissions to the challenge. Potential further work/improvements:
 - ▶ Diversification:
 - ▶ Generate many initial Solutions, then apply local search.
 - ▶ Ideas to swap out multiple critical arcs at once, to avoid getting stuck.
 - ▶ Route Selection: At every step check all routing changes?
 - ▶ Connections:
 - ▶ Soft penalties?
 - ▶ Work with more complex objective functions (penalty for arriving too early)?
 - ▶ Do we miss structure? The method can also cope with ‘Montage/Demontage’ structure.