Online Supplement: An Exact Branch-Price-and-Cut Algorithm for the Unrelated Parallel Machine Scheduling Problem

EC.1. Proof of Statements

EC.1.1. Proof of Proposition 1

Proof. We consider two labels L_1 and L_2, each of which satisfies the three conditions of Proposition 1, i.e., $v(L_1) = v(L_2)$, $C(L_1) \leq C(L_2)$, and $r(L_1) - \sum_{l \in \mathcal{C}: \mathcal{S}(L_1) > \mathcal{S}(L_2)} \mu_l \leq r(L_2)$. We prove the proposition by showing that every feasible completion of L_2 is also feasible for L_1 and yields a label whose reduced cost is not greater than that of the label obtained by applying the same completion to L_1.

For ease of description, let the symbol $\oplus$ denote a function that combines a partial schedule L with a set of additional jobs (an extension) E to produce a complete and feasible schedule $(L \oplus E)$, following the extension rules mentioned in Section 3.1.

Let $E = (v_1, v_2, \ldots, v_g = n+1)$ be any feasible extension of L_2. The feasibility of E for L_2 means that, at each label extension e, the next job v_e is in the set $\alpha(L_2 \oplus v_1 \oplus \ldots \oplus v_{e-1})$. Since $v(L_1) = v(L_2)$, the initial job ordering restrictions are the same for both labels, i.e., $\alpha(L_1) = \alpha(L_2)$. Moreover, because both labels extend along the same sequence E, the state updates identically at each label extension. Therefore, for all e, $\alpha(L_1 \oplus v_1 \oplus \ldots \oplus v_{e-1}) = \alpha(L_2 \oplus v_1 \oplus \ldots \oplus v_{e-1})$, and thus $v_e \in \alpha(L_1 \oplus v_1 \oplus \ldots \oplus v_{e-1})$. This shows that E is also a feasible extension for L_1.

Subsequently, we need to show that $r(L_1 \oplus E) - r(L_2 \oplus E) \leq 0$. The reduced cost of $r(L_i \oplus E)$ (for $i = 1, 2$) can be written as:

$$r(L_i \oplus E) = r(L_i) + \sum_{e=1}^{g-1} w_{v_e} \left(C(L_i) + \sum_{w=1}^{e} \hat{p}_{v_w} \right) - \sum_{e=1}^{g-1} \pi_{v_e} - \sum_{l=1}^{n_S} ndual_i^l \mu_l,$$

which involves terms related to the lm-SRCs, making them difficult to quantify. Therefore, we refer to Algorithm 1 to denote the values of these terms. For convenience, let $ndual^l$ denote the number of times its dual value u_l has been subtracted from the reduced cost for each lm-SRC $l \in \mathcal{C}$ as line 8 in Algorithm 1 is executed.

Thus, the difference between the reduced costs of schedules $L_1 \oplus E$ and $L_2 \oplus E$ is given by:

$$\begin{aligned} & r(L_1 \oplus E) - r(L_2 \oplus E) \\ &= \underbrace{\sum_{e=1}^{g-1} w_{v_e} \left(C(L_1) - C(L_2) \right)}_{\text{(a)}} + \underbrace{\left(r(L_1) - r(L_2) \right) - \sum_{l=1}^{n_S} \left(ndual_1^l - ndual_2^l \right) \mu_l}_{\text{(b)}}. \end{aligned}$$

where the first term (a) is non-positive due to the given condition $C(L_1) \leq C(L_2)$, and the second term (b) depends on the vector *state* $\mathcal{S}(L_i)$, as shown in Algorithm 1. We have two cases for each $l \in \mathcal{C}$:

(1) **Case 1** ($\mathcal{S}(L_1)[l] \leq \mathcal{S}(L_2)[l]$): In this case, according to Algorithm 1, we have $ndual_1^l \leq ndual_2^l$.

(2) **Case 2** ($\mathcal{S}(L_1)[l] > \mathcal{S}(L_2)[l]$): Since $0 \leq \mathcal{S}(L_i)[l] < 1$, we have $\mathcal{S}(L_1)[l] - \mathcal{S}(L_2)[l] < 1$. Based on Algorithm 1, we have $ndual_1^l - 1 \leq ndual_2^l$.

Based on the two cases above and the fact that $\boldsymbol{\mu} \leq 0$, the second term can be rewritten as:

$$r(L_1) - r(L_2) - \sum_{l \in \mathcal{C}: \mathcal{S}(L_1)[l] \leq \mathcal{S}(L_2)[l]} (ndual_1^l - ndual_2^l)\mu_l - \sum_{l \in \mathcal{C}: \mathcal{S}(L_1)[l] > \mathcal{S}(L_2)[l]} (ndual_1^l - ndual_2^l)\mu_l$$
$$\leq r(L_1) - r(L_2) - \sum_{l \in \mathcal{C}: \mathcal{S}(L_1)[l] > \mathcal{S}(L_2)[l]} \mu_l.$$

Therefore, the term is also non-positive due to the condition $r(L_1) - \sum_{l \in \mathcal{C}: \mathcal{S}(L_1) > \mathcal{S}(L_2)} \mu_l \leq r(L_2)$. By adding up all the terms, we conclude that $r(L_1 \oplus E) - r(L_2 \oplus E) \leq 0$. Thus, the proposition is proved. □

EC.1.2. Proof of Theorem 1

Proof. Let $S_k = \{j_1, \ldots, j_{n_k}\}$ denote the sequence of jobs processed on machine $k \in M$, where n_k is the number of jobs processed. A solution S can be represented by a set of schedules $\{S_q\}_{q \in \bar{M}}$, where $\bar{M} \subseteq M$. These schedules satisfy $\bigcup_{q \in \bar{M}} \{S_q\} = N$ and $\bigcap_{q \in \bar{M}} \{S_q\} = \emptyset$, where $q \leq m$. In the following, given a schedule S_k, we denote with $j_h \in S_k$ the job processed on machine k, with completion time $C_{j_h}^k$ and start processing time $ST_{j_h}^k = C_{j_h}^k - p_{j_h k}$. Let $S^* = \{S_q^*\}_{q \in \bar{M}^*}$ denote an optimal solution, where v^* is the index of the last machine to complete its schedule among all machines, i.e., $v^* = \arg\max_{q \in \bar{M}^*} \{C_{j_{n_q}}^q\}$, using the higher index as a tiebreaker the machine with the higher index is selected, and i^* is the index of the last completed job, i.e., $i^* = j_{n_{v^*}}$.

To prove this theorem, we need to show that, considering solution S^*, the completion time $C_{j_h}^k$ of job j_h on machine k needs to satisfy the following conditions: (1) $C_{j_h}^k \leq ub^*$. (2) Additionally, $C_{j_h}^k$ must satisfy $C_{j_h}^k \leq \max_{j' \in \beta j_h k} \left\{ub_{j'}^k\right\} + p_{j_h k}$.

For condition (1), let S' denote a solution that is the same as the optimal solution S^*, except for job i^*. Specifically, job i^* is scheduled to the end of machine v^* in S^*, whereas job i^* is scheduled to the end of a machine other than v^* in S', i.e., to one machine in $M \setminus \{v^*\}$.

Due to the optimality of S^*, the total weighted completion time of S^* is less than or equal to that of S', that is to say

$$w_{i^*}(ST_{i^*}^{v^*} + p_{i^* v^*}) \leq w_{i^*} \left(\sum_{j_h \in S_k^*} p_{j_h k} + p_{i^* k} \right), \quad \forall k \in M/\{v^*\}.$$

where $w_{i^*} > 0$ and $S_k^* = \emptyset$ if $k \notin \bar{M}^*$.

If we divide both sides of the above inequalities by $w_{i^*} > 0$ and rearrange the terms, we obtain:

$$ST_{i^*}^{v^*} \leq \sum_{j_h \in S_k^*} p_{j_h k} + p_{i^* k} - p_{i^* v^*}, \forall k \in M/\{v^*\}. \tag{EC.1}$$

Aggregating the m inequalities (EC.1) and substituting $ST_{i^*}^{v^*} = \sum_{j_h \in S_{v^*}^*} p_{j_h v^*} - p_{i^* v^*}$, we obtain

$$mST_{i^*}^{v^*} \leq \sum_{k \in M} \sum_{j_h \in S_k^*} p_{j_h k} + \sum_{k \in M} p_{i^* k} - (m+1)p_{i^* v^*} \leq$$
$$\sum_{j \in N} \max_{k \in M} \{p_{jk}\} + \sum_{k \in M} p_{i^* k} - (m+1)p_{i^* v^*},$$

which reduces to

$$ST_{i^*}^{v^*} \leq \frac{\sum_{j\in N}\max_{k\in M}\{p_{jk}\}}{m} + \max_{j\in N,k\in M}\{p_{jk}\} - \frac{(m+1)p_{i^*v^*}}{m}.$$

Thus, the completion time $C_{i^*}^{v^*}$ is bounded by

$$C_{i^*}^{v^*} = ST_{i^*}^{v^*} + p_{i^*v^*} \leq \frac{\sum_{j\in N}\max_{k\in M}\{p_{jk}\}}{m} + \max_{j\in N,k\in M}\{p_{jk}\} = ub^*.$$

Since job i^* is the last completed job in the optimal solution S^*, for any job j_h on machine $k \in \bar{M}^*$, $C_{j_h}^k \leq ub^*$. Thus, the condition is proved.

We will prove condition (2) by induction. For any machine $k \in M$, we start with the base case, i.e., the first job j_1 in SWPT^k order, and thus the job ordering restriction $\beta_{j_1}^k = \{0\}$. Based on lines 12-18 of Algorithm 2, we can obtain that $ub_{j_1}^k = p_{j_1k}$. In addition, we can easily compute its completion time $C_{j_1}^k = p_{j_1k}$. This confirms the theorem for j_1 since $C_{j_1}^k = p_{j_1k} = ub_{j_1}^k$.

For the inductive step, suppose the theorem holds for jobs $j_1, ...j_h, ..., j_{n_k-1}$ such that $C_{j_h}^k \leq ub_{j_h}^k$. Without loss of generality, we assume that in the optimal schedule, job $j_h \in \beta_{j_{n_k}}^k$ precedes job j_{n_k} and they are consecutive. Thus, the completion time of job j_{n_k} is given by $C_{j_{n_k}}^k = C_{j_h}^k + p_{j_{n_k}k}$. In calculating $ub_{j_{n_k}}^k$, based on Algorithm 2, there are two cases when line 18 of the algorithm is reached:

(1) $maxPreJ_{j_{n_k}}^k = j_h$. In this case, we compute $ub_{j_{n_k}}^k$ as follows:

$$\begin{aligned} ub_{j_{n_k}}^k &= ub_{j_h}^k + p_{j_{n_k}k} \\ &\geq C_{j_h}^k + p_{j_{n_k}k} \\ &= C_{j_{n_k}}^k. \end{aligned}$$

Thus, the theorem holds for case (1), meaning $C_{j_{n_k}}^k \leq ub_{j_{n_k}}^k$.

(2) $maxPreJ_{j_{n_k}}^k \neq j_h$. In this case, based on lines 13-17 of Algorithm 2, we can easily obtain that $ub_{maxPreJ_{j_{n_k}}}^k \geq ub_{j_h}^k$. Then, it is established, according to line 18, that

$$\begin{aligned} ub_{j_{n_k}}^k &= ub_{maxPreJ_{j_{n_k}}}^k + p_{j_{n_k}k} \\ &\geq ub_{j_h}^k + p_{j_{n_k}k} \\ &\geq C_{j_h}^k + p_{j_{n_k}k} \\ &= C_{j_{n_k}}^k \end{aligned}$$

Hence, the theorem also holds for case (2), indicating $C_{j_{n_k}}^k \leq ub_{j_{n_k}}^k$.

In conclusion, $C_{j_h}^k \leq ub_{j_h}^k = \max_{j'\in\beta_{j_h}^k}\left\{ub_{j'}^k\right\} + p_{j_hk}$. Hence, the condition is proved. Considering that conditions (1) and (2) hold, the theorem is proved. □

EC.1.3. Proof of Theorem 2

Proof. We will prove this theorem by contradiction. Assume that removing the arc (j_{h-1}, j_h) from the directed acyclic graph G^k, where $j_{h-1} = \arg\max_{j'\in\beta_{j_h}^k} ub_{j'}^k$, does not change the upper bound, i.e., $ub_{j_h}^k =$

Algorithm 5: Calculate the Coefficient of Schedule S_s on Machine k.

```
Input: C, M^k, ρ, s, k
Output: ω
begin
    ω ← 0, state ← 0;
    foreach job j in schedule S_s on machine k (in order) do
        if j ∉ M^k then
            state ← 0;
        end
        else if j ∈ C then
            state ← state + ρ;
            if state ≥ 1 then
                ω ← ω + 1, state ← state − 1;
            end
        end
    end
    return ω;
end
```

$ub^k_{j_{h-1}} + p_{j_h k}$. Under this assumption, it follows that the job ordering restriction $\beta^k_{j_h}$ equals $\beta^k_{j_h}/\{j_{h-1}\}$. Referring to lines 12-18 of Algorithm 2, it is evident that $maxPreJ^k_{j_h} \neq j_{h-1}, maxPreJ^k_{j_h} \in \beta^k_{j_h}$. Hence

$$ub^k_{j_h} = ub^k_{maxPreJ^k_{j_h}} + p_{j_h k} \neq ub^k_{j_{h-1}} + p_{j_h k}.$$

This contradicts the assumption that $ub^k_{j_h} = ub^k_{j_{h-1}} + p_{j_h k}$, thereby proving the theorem.

It is easy to see that $ub^k_{j_{h-1}} \geq ub^k_{j'}$ for all $j' \in \beta^k_{j_h}$, that is, $ub^k_{j_{h-1}} \geq ub^k_{maxPreJ_{j_h}}$. Consequently, removing the arc (j_{h-1}, j_h) from the directed acyclic graph G^k results in a decrease (or at least no increase) in $ub^k_{j_h}$.

□

EC.2. Additional Algorithm Details

This section illustrates additional details about the BPC algorithm, including the use of additional cuts (§EC.2.1), the neighborhood search heuristic pricing algorithm (§EC.2.2), the backward labeling algorithm (§EC.2.3), and the variable fixing by reduced costs method (§EC.2.4).

EC.2.1. Additional Cuts

The procedure for calculating the coefficient ω is shown in Algorithm 5. If $\mathcal{M}^k = N$, the procedure returns a value of ω equal to $\left\lfloor \rho \sum_{j \in C} a^k_{js} \right\rfloor$, making the lm-SRC equivalent to the SRC. However, if $\mathcal{M}^k \subset N$, it may occur that S_s visits a job $j \notin \mathcal{M}^k$ when $state > 0$, causing *state* to reset to zero. In this case, the returned coefficient may be smaller than $\left\lfloor \rho \sum_{j \in C} a^k_{js} \right\rfloor$.

We apply the separation strategy for lm-SRCs as follows. Firstly, we enumerate all triples of jobs within N. For each triple C, we verify whether it satisfies the inequality (5). The (C, ρ)-SRC is violated if this inequality is unmet. Subsequently, we determine a memory set $\mathcal{M}^k$ for each machine $k \in M$. Let y^k_s be a fractional solution associated with the LP relaxation of the RMP, where $k \in M$ and $s \in \Omega_k$. The goal is to obtain small memory sets $\{\mathcal{M}^1, \ldots, \mathcal{M}^k\}$ such that the coefficients of all variables y^k_s with positive values are the same in both the SRC and the lm-SRCs. In this way, the lm-SRC has the same violation as the SRC. The algorithm for generating $\mathcal{M}^k$ is presented in Algorithm 6.

Algorithm 6: Calculate the Memory Set on Machine k.

Input: C, ρ, y_s^k
Output: $\mathcal{M}^k$
begin
$\mathcal{M}^k \leftarrow C$;
for *each schedule* S_s *on machine* k *such that* $y_s^k > 0$ *and* $\lfloor \rho \sum_{j \in C} a_{js}^k \rfloor > 0$ **do**
$state \leftarrow 0$, $Aux \leftarrow \emptyset$;
for *every job* j *in schedule* S_s *on machine* k *(in order)* **do**
if $j \in C$ **then**
$state \leftarrow state + \rho$;
if $state \geq 1$ **then**
$\mathcal{M}^k \leftarrow \mathcal{M}^k \cup Aux$, $Aux \leftarrow \emptyset$, $state \leftarrow state - 1$;
end
else if $state \geq 0$ **then**
$Aux \leftarrow Aux \cup \{j\}$;
end
end
end
end
return $\mathcal{M}^k$;
end

The lm-SRCs have a reduced impact on pricing compared to SRCs when their memory sets are small. When an SRC is added, its dual variable may weaken the dominance rule (9) in all buckets. As additional SRCs are separated, this can quickly lead to an exponential proliferation of non-dominated labels (Pecin et al. 2017b). The lm-SRC with a small memory has significantly less impact, as it weakens dominance only in the buckets of the memory set $\mathcal{M}^k$.

EC.2.2. Neighborhood Search Heuristic Pricing Algorithm

Algorithm 7: Heuristic Extension.

Input: L', B, Γ, $\boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho$
Output: B, Γ
begin
while *there exists bucket arc* $\gamma \in \Gamma$ *such that* $b_\gamma = b^{L'}$ **do**
Randomly select a bucket arc γ^* such that $b_{\gamma^*} = b^{L'}$;
if *Label Extension*$(L', \gamma^*, L, \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho)$ *(see Algorithm 4)* **then**
Insert in b^L, $L' \leftarrow L$;
end
end
return B, Γ;
end

The detailed procedure for the neighborhood search heuristic pricing is presented in Algorithm 8, which utilizes Algorithm 7 to extend forward labels in a heuristic manner. In Algorithm 7, instead of traversing all the bucket arcs, forward labels are extended along randomly selected bucket arcs (in line 2). This process is repeated until a complete forward label is obtained. The neighborhood search heuristic pricing begins with an initial forward label L^{init}, which is extended by Algorithm 7 to produce an initial complete label L^{best} (in lines 1-5). The label heuristic extension process requires additional records of the visited buckets $\{b_0, \ldots, b_g\}$ (in line 6). Subsequently, it generates the neighbor label $L^{neighbor}$ based on the label and the buckets. Firstly, a bucket b_r is randomly selected from the visited buckets $\{b_0, \ldots, b_g\}$ (in line 8). By visiting buckets $\{b_0, \ldots, b_r\}$, a partial neighbor label $L^{neighbor}$ is generated (lines 9-13). Then, the partial neighbor

label $L^{neighbor}$ is extended using Algorithm 7 to produce a complete neighbor label (in line 14). Each iteration produces $numNeighbors$ neighbor labels, and the algorithm continues until $maxIterations$ iterations are reached.

Algorithm 8: Neighborhood Search Heuristic Pricing Algorithm.

Input: $B, \Gamma \cup \Psi, \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho$
Output: S^{best}
begin
1 $L^{init} \leftarrow (0, \emptyset, 0, 0, \emptyset, \{1, \ldots, n\})$ and insert it to its bucket;
2 Run Algorithm 7, *Heuristic Extension* ($L^{init}, B, \Gamma, \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho$);
3 $iter \leftarrow 0$;
4 **while** $iter < maxIterations$ **do**
5 $L^{best} \leftarrow \arg\min_{L \text{ where } v(L)=n+1} r(L)$;
6 $\{b_0, b_1, \ldots, b_g\} \leftarrow$ Buckets visited by label L^{best};
7 **for** $num = 1$ **to** $numNeighbors$ **do**
8 Randomly select a bucket b_r from $\{b_0, \ldots, b_g\}$;
9 $L^{neighbor} \leftarrow (0, \emptyset, 0, 0, \emptyset, \{1, \ldots, n\})$;
10 **foreach** *bucket arc* $\gamma = (b_i, (j_{b_i}, j_{b_{i+1}}), b_{i+1}), i = 0, \ldots, r-1$ **do**
11 Label Extension($L^{neighbor}, \gamma, L, \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho$) (see Algorithm 4) and insert in b^L;
12 $L^{neighbor} \leftarrow L$;
13 **end**
14 Run Algorithm 7, *Heuristic Extension* ($L^{neighbor}, B, \Gamma, \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathcal{M}, \rho$);
15 **end**
16 $iter \leftarrow iter + 1$;
17 **end**
18 $S^{best} = \arg\min_{S^L \text{ where } v(L)=n+1} r(L)$;
19 **return** $B, \Gamma \cup \Psi, S^{best}$;
20 **end**

EC.2.3. Exact Backward Labeling Algorithm

This section describes the backward labeling algorithm. We refer to the labels and schedules defined in Section 3.1 as *forward labels* L and *forward schedules* S. In contrast, a *backward schedule* $\overleftarrow{S}$ (associated with a *backward label* $\overleftarrow{L}$) begins at a job $i \in N$ and terminates at sink node $n+1$; it is complete if $i = 0$. For brevity, throughout this section, we omit the machine index k and denote the processing time for job j as $\hat{p}_j$, where $\hat{p}_j = p_{jk}$.

In a backward labeling fashion, generated labels are initialized with node $n+1$ with a completion time equal to the time horizon H. Consequently, a backward path reaching source node 0 implicitly defines a complete schedule spanning the interval $[t_0, H]$, where t_0 denotes the start time. If $t_0 > 0$, idle time exists within the interval $[0, t_0]$. In the context of the PSP, such a delay increases the total weighted completion time and, consequently, the reduced cost. This observation also applies to partial backward schedules: shifting a backward schedule to an earlier start time decreases its reduced cost, a property formalized in the following proposition.

PROPOSITION EC.1 **(Left-shift).** *Let* $\overleftarrow{S} = (j_0, \ldots, j_p = n+1)$ *be a feasible backward schedule spanning the time interval* $[t, H]$, *with* $t > 0$.

(i) There exists a left-shifted schedule $\overleftarrow{S}'$, *starting at time* t', *with* $0 \leq t' < t$, *that remains feasible.*

(ii) Let $q(\overleftarrow{S}) = \sum_{j \in \overleftarrow{S}} w_j$ *denote the* cumulative weight *of the jobs in* $\overleftarrow{S}$*. The reduced cost* $r(\overleftarrow{S}, t)$ *of schedule* $\overleftarrow{S}$ *when starting at time* t *can be computed as*

$$r(\overleftarrow{S}, t) = r(\overleftarrow{S'}, t') + (t - t') \cdot q(\overleftarrow{S}), \tag{EC.2}$$

where $r(\overleftarrow{S'}, t')$ *is the reduced cost of schedule* $\overleftarrow{S'}$ *starting at* t'*. Since* $q(\overleftarrow{S}) > 0$ *and* $t - t' > 0$*, it follows that* $r(\overleftarrow{S}', t') < r(\overleftarrow{S}, t)$*, and therefore left-shifting the schedule strictly reduces the reduced cost.*

Proof. Let $\Delta = t - t'$ denote the magnitude of the time shift. Given that $0 \leq t' < t$, it follows that $\Delta > 0$. For any job j in the sequence $\overleftarrow{S}$, let $ST_j(t)$ and $C_j(t)$ denote its start time and completion time, respectively, assuming the schedule starts at time t. Note that $C_j(t) = ST_j(t) + \hat{p}_j$. The left-shifted schedule $\overleftarrow{S}'$ is constructed by advancing the timing of every job by exactly Δ. Specifically, the modified start and completion times are defined as $ST'_j(t') = ST_j(t) - \Delta$ and $C'_j(t') = C_j(t) - \Delta$.

(i) We first establish the feasibility of the left-shifted schedule $\overleftarrow{S}'$. Similar to forward schedules, the backward schedule is considered *feasible* if all jobs contained within it satisfy the specified time interval imposed on the start times. Let $\overleftarrow{lb}_j$ and $\overleftarrow{ub}_j$ denote the lower and upper bounds, respectively, for the start time of job j. We derive the start time bounds from the completion time bounds $[\hat{p}_j, dsub_j]$ defined in Section 3.2, i.e., $\overleftarrow{lb}_j = 0$ and $\overleftarrow{ub}_j = dsub_j - \hat{p}_j$.

Given that the original schedule $\overleftarrow{S}$ is feasible, it holds that $ST_j(t) \leq dsub_j - \hat{p}_j$. Since $\Delta > 0$, it follows that $ST'_j(t') < ST_j(t) \leq dsub_j - \hat{p}_j$, thereby satisfying the upper bound.The shifted schedule $\overleftarrow{S}'$ begins at $t' \geq 0$. Consequently, the start time of any job j must satisfy $ST'_j(t') \geq t' \geq 0$. This ensures that the lower bound is maintained. Thus, the feasibility of the left-shifted schedule is preserved.

(ii) Next, we analyze the reduced cost. The reduced cost of the schedule $\overleftarrow{S}$ at start time t is:

$$r(\overleftarrow{S}, t) = \sum_{r=0}^{p} (w_{j_r} C_{j_r}(t) - \pi_{j_r}) - \sum_{l=1}^{n_S} ndual^l(\overleftarrow{S})\mu_l$$

where $ndual^l(\overleftarrow{S})$ denotes the number of times the dual value μ_l is subtracted (based on the state of the lm-SRCs) along the schedule $\overleftarrow{S}$. Note that the dual contribution depends solely on the set of visited jobs and their sequence.

By substituting $C_{j_r}(t)$, $r = 0, \ldots, p$, with the shifted completion times $C'_{j_r}(t') = C_{j_r}(t) - \Delta$, and considering that $\Delta = (t - t')$ and $q(\overleftarrow{S}) = \sum_{r=0}^{p} w_{j_r}$, we obtain the reduced cost of the left-shifted schedule $\overleftarrow{S}'$ starting at time t':

$$\begin{aligned} r(\overleftarrow{S}', t') &= \sum_{r=0}^{p} (w_{j_r} \cdot (C_{j_r}(t) - \Delta) - \pi_{j_r}) - \sum_{l=1}^{n_S} ndual^l(\overleftarrow{S})\mu_l \\ &= \sum_{r=0}^{p} (w_{j_r} \cdot C_{j_r}(t) - \pi_{j_r}) - \sum_{l=1}^{n_S} ndual^l(\overleftarrow{S})\mu_l - \sum_{r=0}^{p} w_{j_r} \cdot \Delta \\ &= r(\overleftarrow{S}, t) - (t - t') \cdot q(\overleftarrow{S}). \quad \square \end{aligned}$$

Building upon Proposition EC.1, we consider the boundary case where $t' = 0$. We define this left-shifted schedule as the *baseline schedule* and its reduced cost as the *baseline reduced cost*, denoted by $\bar{r}(\overleftarrow{S})$. By setting $t' = 0$ in Equation (EC.2), the reduced cost of any backward schedule $\overleftarrow{S}$ starting at time t can be reformulated as:

$$r(\overleftarrow{S}, t) = \bar{r}(\overleftarrow{S}) + t \cdot q(\overleftarrow{S}). \tag{EC.3}$$

where $\bar{r}(\overleftarrow{S})$ and $q(\overleftarrow{S})$ are constant values determined solely by the structure of $\overleftarrow{S}$, making $r(\overleftarrow{S}, t)$ an affine function of t. This function is valid for any start time t such that the schedule completes within the time horizon ($C_{j_p}(t) \leq H$).

Equation (EC.3) provides a clear physical interpretation: any backward schedule $\overleftarrow{S}$ starting at time t can be viewed as the baseline schedule shifted right by t units. Consequently, the reduced cost of $\overleftarrow{S}$ comprises the baseline reduced cost $\bar{r}(\overleftarrow{S})$ and a linear penalty $t \cdot q(\overleftarrow{S})$ incurred by the shift. To evaluate the reduced cost of a backward schedule for any start time t, we need only maintain two parameters: the baseline reduced cost $\bar{r}$ and the cumulative weight q. This observation forms the basis for our backward label definition.

EC.2.3.1. Backward Label Definition, Extension, and Dominance Rules We refer to a tuple $\overleftarrow{L} = (v, \mathcal{V}, \bar{r}, q, d, \mathcal{S}, \beta)$, representing a partial backward schedule, as a *backward label*. In this tuple: v denotes the first job in the schedule; $\mathcal{V}$ represents the set of jobs contained in the schedule; $\bar{r}$ is the baseline reduced cost; q is the cumulative weight; d is the duration of the schedule. Since the backward extension initializes at the horizon H, the value $H - d$ represents the start time of the schedule assuming it completes at H; $\mathcal{S}$ is the vector of states corresponding to the lm-SRCs; and β represents the job ordering restriction, i.e., the set of jobs that can be scheduled before v. The initial backward label is denoted by $\overleftarrow{L}^{init} = (n+1, \emptyset, 0, 0, 0, \mathbf{0}, \{1, \ldots, n\})$.

Let $\overleftarrow{L}'$ be the backward label obtained by adding job i to the head of the partial schedule represented by $\overleftarrow{L}$ corresponding to traversing arc $(i, v(\overleftarrow{L}))$. Note that job i must belong to the set $\beta(\overleftarrow{L})$. The extension rule for $\overleftarrow{L}$ is defined as follows: (i) $v(\overleftarrow{L}') = i$; (ii) $\mathcal{V}(\overleftarrow{L}') = \mathcal{V}(\overleftarrow{L}) \cup \{i\}$; (iii) $q(\overleftarrow{L}') = q(\overleftarrow{L}) + w_i$; (iv) $d(\overleftarrow{L}') = d(\overleftarrow{L}) + \hat{p}_i$. (v) The baseline reduced cost $\bar{r}(\overleftarrow{L}')$ is updated based on the affine property derived in Equation EC.3. Recall that the baseline reduced cost $\bar{r}$ is defined as the reduced cost of the partial schedule shifted to start at time $t = 0$. Therefore, job i starts at 0 and completes at $\hat{p}_i$, and the subsequent schedule $\overleftarrow{L}$ is shifted to start at time $\hat{p}_i$. The update is calculated as:

$$\begin{aligned} \bar{r}(\overleftarrow{L}') &= (w_i \hat{p}_i - \pi_i) + (\bar{r}(\overleftarrow{L}) + \hat{p}_i \cdot q(\overleftarrow{L})) \\ &= \bar{r}(\overleftarrow{L}) + \hat{p}_i \cdot q(\overleftarrow{L}') - \pi_i. \end{aligned} \tag{EC.4}$$

(vi) $\mathcal{S}(\overleftarrow{L}')$ is updated according to the standard lm-SRC logic (see Algorithm 1). Note that if adding job i triggers an lm-SRC dual contribution, the corresponding dual values are subtracted from $\bar{r}(\overleftarrow{L}')$. (vii) $\beta(\overleftarrow{L}') = \beta_i$.

To reduce the number of labels generated, we apply dominance rules. When comparing two backward labels $\overleftarrow{L}_1$ and $\overleftarrow{L}_2$ associated with the same job v, we consider an arbitrary feasible start time t (determined by the completion time of a preceding forward schedule). Based on Proposition EC.1, both backward schedules can be shifted to be aligned at the same start time t. By leveraging the affine structure of the reduced cost function $r(\overleftarrow{L},t)=\bar{r}(\overleftarrow{L})+t\cdot q(\overleftarrow{L})$, checking the dominance of the intercept $\bar{r}$ and the slope q is sufficient to guarantee that $r(\overleftarrow{L}_1,t)\le r(\overleftarrow{L}_2,t)$ holds for any feasible start time t. Therefore, backward dominance rules are given in Proposition EC.2.

PROPOSITION EC.2. *Let $\overleftarrow{L}_1$ and $\overleftarrow{L}_2$ be two backward labels with $v(\overleftarrow{L}_1)=v(\overleftarrow{L}_2)$. If*

(i) $d(\overleftarrow{L}_1)\le d(\overleftarrow{L}_2)$,

(ii) $q(\overleftarrow{L}_1)\le q(\overleftarrow{L}_2)$, *and*

(iii) $\bar{r}(\overleftarrow{L}_1)-\sum_{l\in\mathcal{C}:\mathcal{S}(\overleftarrow{L}_1)[l]>\mathcal{S}(\overleftarrow{L}_2)[l]}\mu_l\le\bar{r}(\overleftarrow{L}_2)$,

with at least one inequality being strict, then label $\overleftarrow{L}_1$ dominates label $\overleftarrow{L}_2$, and thus $\overleftarrow{L}_2$ can be discarded.

Proof. Let $\overleftarrow{L}_1$ and $\overleftarrow{L}_2$ be two backward labels with $v(\overleftarrow{L}_1)=v(\overleftarrow{L}_2)$, satisfying conditions (i)-(iii). We prove the proposition by showing that every feasible completion of $\overleftarrow{L}_2$ is also feasible for $\overleftarrow{L}_1$ and yields a label whose reduced cost is not greater than that of the label obtained by applying the same completion to $\overleftarrow{L}_1$.

Let the symbol $\oplus$ denote a function that combines a partial backward schedule $\overleftarrow{L}$ with a set of additional jobs (an extension) $E=(v_g=0,\dots,v_2,v_1)$, with a total duration $\sum_{r=1}^{g}\hat{p}_{v_r}=t_E$, to produce a complete and feasible schedule $(E\oplus\overleftarrow{L})$, following the backward extension rules. Figure EC.1 provides a visualization of this process. Suppose E is a feasible backward extension for $\overleftarrow{L}_2$; this implies that $H-d(\overleftarrow{L}_2)-t_E>0$ and $v_1\in\beta(\overleftarrow{L}_2)$. Since $d(\overleftarrow{L}_1)\le d(\overleftarrow{L}_2)$ and $v(\overleftarrow{L}_1)=v(\overleftarrow{L}_2)$, it follows that $H-d(\overleftarrow{L}_1)-t_E>0$ and $v_1\in\beta(\overleftarrow{L}_1)=\beta(\overleftarrow{L}_2)$. Consequently, E is also a feasible backward extension for $\overleftarrow{L}_1$ (see Figure EC.1(a)).

Next, we show that $r(E\oplus\overleftarrow{L}_1,0)-r(E\oplus\overleftarrow{L}_2,0)\le 0$. In other words, the reduced cost of any complete schedule constructed by extending $\overleftarrow{L}_1$ is less than or equal to that constructed by extending $\overleftarrow{L}_2$. Note that we evaluate the reduced cost at $t=0$ because Proposition EC.1 guarantees that any feasible backward schedule can be left-shifted to start at $t=0$ to minimize its reduced cost (see Figure EC.1(b)).

Based on Equation (EC.3), the reduced cost $r(E\oplus\overleftarrow{L}_i,0)$ for $i\in\{1,2\}$ is defined as:

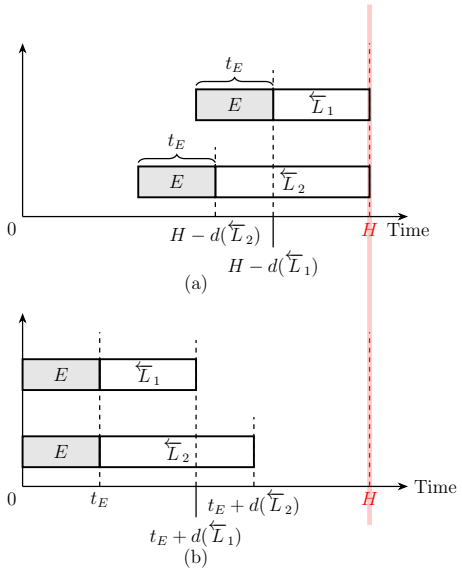
$$\begin{aligned} r(E\oplus\overleftarrow{L}_i,0)&=\bar{r}(E\oplus\overleftarrow{L}_i)+0\cdot q(E\oplus\overleftarrow{L}_i)\\ &=\bar{r}(\overleftarrow{L}_i)+\sum_{r=1}^{g}\hat{p}_{v_r}\cdot\left(q(\overleftarrow{L}_i)+\sum_{k=1}^{r}w_{v_k}\right)-\sum_{r=1}^{g}\pi_{v_r}-\sum_{l=1}^{n_S}ndual_i^l\mu_l, \end{aligned} \tag{EC.5}$$

where $ndual_i^l$ denotes the number of times the dual value μ_l is subtracted from the reduced cost for each lm-SRC $l\in\mathcal{C}$ during the extension.

Thus, the difference between the reduced costs of schedules $(E\oplus\overleftarrow{L}_1)$ and $(E\oplus\overleftarrow{L}_2)$ is given by:

$$r(E\oplus\overleftarrow{L}_1,0)-r(E\oplus\overleftarrow{L}_2,0)$$

Figure EC.1 Illustration of Backward Dominance Rules.



$$= (\bar{r}(\overleftarrow{L}_1) - \bar{r}(\overleftarrow{L}_2)) + \sum_{r=1}^{g} \hat{p}_{v_r} \cdot (q(\overleftarrow{L}_1) - q(\overleftarrow{L}_2)) - \sum_{l=1}^{n_S} (ndual_1^l - ndual_2^l) \cdot \mu_l. \tag{EC.6}$$

Since $q(\overleftarrow{L}_1) \leq q(\overleftarrow{L}_2)$ and $\hat{p}_{v_r} > 0$, the term $\sum_{r=1}^{g} \hat{p}_{v_r} \cdot (q(\overleftarrow{L}_1) - q(\overleftarrow{L}_2))$ is non-positive. Furthermore, given condition (3): $\bar{r}(\overleftarrow{L}_1) - \sum_{l \in \mathcal{C}: \mathcal{S}(\overleftarrow{L}_1)[l] > \mathcal{S}(\overleftarrow{L}_2)[l]} \mu_l \leq \bar{r}(\overleftarrow{L}_2)$, and following an argument identical to that used in the proof of the forward dominance rules, noting that lm-SRC dual contributions depend only on the state vector $\mathcal{S}$. Therefore, we deduce that $(\bar{r}(\overleftarrow{L}_1) - \bar{r}(\overleftarrow{L}_2)) - \sum_{l=1}^{n_S} (ndual_1^l - ndual_2^l) \cdot \mu_l \leq 0$.

Therefore, $r(E \oplus \overleftarrow{L}_1, 0) - r(E \oplus \overleftarrow{L}_2, 0) \leq 0$. Thus, the proposition is proved. □

EC.2.3.2. Bucket Graph-based Backward Labeling Algorithm Analogous to the forward labeling algorithm, we design the backward labeling algorithm on graph $\overleftarrow{G} = (V, \overleftarrow{A})$, with source node $n+1$ and sink node 0. The set $\overleftarrow{A}$ consists of a backward arc $\overleftarrow{a} = (j', j)$ for each corresponding forward arc $\overrightarrow{a} = (j, j') \in \overrightarrow{A}$. Since the forward graph is a directed acyclic graph, the backward graph $\overleftarrow{G}$, formed by reversing all arcs, remains acyclic.

To construct the backward bucket graph, we first determine the lower bound $\overleftarrow{lb}_j$ and upper bound $\overleftarrow{ub}_j$ of the start time for each job j. Recall that for the forward graph, the completion time window for job j is $[lb_j = \hat{p}_j, dsub_j]$, where lb_j represents the lower bound and $dsub_j$ the dynamic shrink upper bound. For the backward graph, we define the start time window $[\overleftarrow{lb}_j, \overleftarrow{ub}_j]$ for each job $j \in N$ by shifting the completion time bounds by the processing time $\hat{p}_j$, i.e., $[\overleftarrow{lb}_j, \overleftarrow{ub}_j] = [0, dsub_j - \hat{p}_j]$.

The backward bucket graph is then constructed based on the start time window $[\overleftarrow{lb}_j, \overleftarrow{ub}_j]$. Let $(\overleftarrow{B}, \overleftarrow{\Gamma} \cup \overleftarrow{\Psi})$ denote the backward bucket graph. For each job $j \in N$, the corresponding set of backward buckets is defined by three parameters: a step size $\overleftarrow{d}_j$, the lower bound $\overleftarrow{lb}_j$, and the upper bound $\overleftarrow{ub}_j$. We define $\overleftarrow{I}_j = \{0, 1, \ldots, R_j - 1\}$ as the index set for the backward buckets of job j, where R_j is the predefined number of buckets. Consequently, the step size is calculated as $\overleftarrow{d}_j = (\overleftarrow{ub}_j - \overleftarrow{lb}_j)/R_j$.

Each backward bucket $\overleftarrow{b} \in \overleftarrow{B}$ is represented by a pair consisting of a job and an upper bound, $\left(j_{\overleftarrow{b}}, \overleftarrow{ub}_{\overleftarrow{b}}\right)$, where $j_{\overleftarrow{b}} = j$ and $\overleftarrow{ub}_{\overleftarrow{b}} = \overleftarrow{ub}_j - i \cdot \overleftarrow{d}_j$, with $j \in N$ and $i \in \overleftarrow{I}_j$. A backward label $\overleftarrow{L}$ is contained in bucket $\overleftarrow{b}$ if $v(\overleftarrow{L}) = j_{\overleftarrow{b}}$ and the start time satisfies $\overleftarrow{ub}_{\overleftarrow{b}} \geq H - d(\overleftarrow{L}) > \overleftarrow{ub}_{\overleftarrow{b}} - \overleftarrow{d}_{j_{\overleftarrow{b}}}$.

Each backward bucket arc $\overleftarrow{\gamma} \in \overleftarrow{\Gamma}$ is defined by a tuple $(\overleftarrow{b}_{\text{tail}}, \overleftarrow{a}_{\overleftarrow{\gamma}}, \overleftarrow{b}_{\text{head}})$, representing two buckets and an underlying backward arc. The set $\overleftarrow{\Gamma}$ contains a backward bucket arc if the two buckets satisfy the following condition:

$$\overleftarrow{ub}_{\overleftarrow{b}_{\text{head}}} \geq \min\{\overleftarrow{ub}_{\overleftarrow{b}_{\text{tail}}} - \hat{p}_{j_{\overleftarrow{b}_{\text{head}}}}, \overleftarrow{ub}_{j_{\overleftarrow{b}_{\text{head}}}}\} > \overleftarrow{ub}_{\overleftarrow{b}_{\text{head}}} - \overleftarrow{d}_{j_{\overleftarrow{b}_{\text{head}}}}.$$

The set $\overleftarrow{\Psi}$ contains backward bucket arcs $(\overleftarrow{b}, \overleftarrow{b}')$ such that $\overleftarrow{b} \in \overleftarrow{B}$ and $\overleftarrow{b}' \in \Phi_{\overleftarrow{b}}$, where $\Phi_{\overleftarrow{b}}$ is the set of adjacent component-wise smaller buckets of $\overleftarrow{b}$.

The procedure of the backward labeling algorithm is similar to that of the forward labeling algorithm; therefore, additional details are omitted for the sake of brevity.

EC.2.4. Variable Fixing by Reduced Costs

This section describes the variable-fixing procedure used to reduce the size of the graph G. For notational simplicity, the machine index k is omitted. We follow the scheme of the variable fixing method outlined in Bulhões et al. (2020).

Let $(\boldsymbol{\pi}, \boldsymbol{\sigma}, \boldsymbol{\mu})$ be a feasible dual solution of the MP, with a lower bound LB as the value of the dual bound and an upper bound UB as the value of the incumbent solution. The primal-dual gap is defined as $UB - LB$. An arc $a = (i, j) \in A$ can be removed from the graph if the reduced cost of the schedule with the minimum reduced cost among all schedules containing the arc a, denoted as $r_{a\min}$, is larger than the gap $UB - LB$.

To compute $r_{a\min}$ efficiently, we utilize the sets of non-dominated labels generated by the mono-directional labeling algorithms. Let $\mathcal{F}(i)$ denote the set of non-dominated forward labels ending at node i, and let $\mathcal{B}(j)$ denote the set of non-dominated backward labels starting at node j. Once both labeling algorithms have been executed for the full time horizon, the minimum reduced cost $r_{a\min}$ for any arc $a = (i, j)$ is obtained by evaluating the feasible concatenations between labels in $\mathcal{F}(i)$ and $\mathcal{B}(j)$. This operation effectively connects the forward schedule associated with $L \in \mathcal{F}(i)$ and the backward schedule associated with $\overleftarrow{L} \in \mathcal{B}(j)$ via the arc $a = (i, j)$. A concatenation is considered feasible only if the following conditions are met (i) the set of visited jobs must be disjoint, i.e., $j \in \alpha(L)$; (ii) the completion time of the forward label must not exceed the latest possible start time of the backward label, i.e., $C(L) \leq H - d(\overleftarrow{L})$.

A challenge arises as the reduced cost of a backward label depends on its start time. Based on Proposition EC.1, the two partial schedules are concatenated without idle time to minimize the resulting reduced cost, implying the backward label is left-shifted to start at $C(L)$. By leveraging the affine structure (EC.3), we evaluate the reduced cost of the complete schedule $L \oplus \overleftarrow{L}$ starting at $t = 0$ as follows:

$$r(L \oplus \overleftarrow{L}, 0) = r(L, 0) + r(\overleftarrow{L}, C(L)) - \sum_{l \in \mathcal{C}: \mathcal{S}(L)[l] + \mathcal{S}(\overleftarrow{L})[l] \geq 1} \mu_l$$

$$= r(L,0) + \left[\bar{r}(\overleftarrow{L}) + C(L)\cdot q(\overleftarrow{L})\right] - \sum_{l\in\mathcal{C}:\mathcal{S}(L)[l]+\mathcal{S}(\overleftarrow{L})[l]\geq 1} \mu_l, \tag{EC.7}$$

where the states $\mathcal{S}(L)$ and $\mathcal{S}(\overleftarrow{L})$ are considered in the last term. The reduced cost of the resulting complete label is adjusted if $\mathcal{S}(L)[l] + \mathcal{S}(\overleftarrow{L})[l] \geq 1$, for $l \in \mathcal{C}$.

The minimum reduced cost $r_{a\min}$ associated with arc $a = (i,j)$ is determined by minimizing over all feasible label pairs:

$$r_{a\min} = \min_{\{L\in\mathcal{F}(i),\overleftarrow{L}\in\mathcal{B}(j):(i),(ii)\}} \left\{r(L\oplus\overleftarrow{L},0)\right\}, \tag{EC.8}$$

and if $r_{a\min} \geq (UB - LB)$, arc $(i,j) \in A$ can be safely removed.

EC.3. Additional Numerical Experiments

Tables EC.1 and EC.2 demonstrate that each component contributes to improving the BPC's performance. In particular, it shows that the dominance rules play the most important role because they allow solving 298 more instances than the BPC algorithm with this feature disabled. Next, heuristic pricing is relatively important because seven fewer instances are solved when heuristic pricing is disabled. Furthermore, while disabling lm-SRCs separation results in solving just two fewer instances, its impact on the BPC algorithm is far more significant. To provide a more comprehensive view, Table EC.2 also reports the average computation time of the BPC algorithm ("BPC Time [s]") and the percentage increase in the algorithm time relative to the BPC time ("Inc. [%]") under the heading "No Lm-SRC". The computation time increases significantly after disabling lm-SRCs separation. Notably, for medium-scale instances, the increase exceeds twofold. For example, for instances with eight machines and 100 jobs, the average computation time rose from 96.01 seconds to 270.43 seconds, representing a 181.67% increase. Finally, these tables also show that other components play a similarly vital role in the number of instances solved to optimality.

EC.4. Order Picking in Warehouse System at JD.com

This section briefly outlines the practical application that inspired our study of the UPMSP.

The warehouse system at JD.com significantly differs from traditional picker-to-parts systems by adopting a parts-to-picker system, enhanced by an extensive network of sensors and devices, including barcode scanners and Automated Guided Vehicles (AGVs). In this system, pickers no longer need to walk to shelves to retrieve the corresponding Stock Keeping Units (SKUs) and return. Instead, AGVs transport unit shelves to *stations* staffed by pickers, where pickers perform SKUs—referred to as *tasks*. The time taken for these tasks is referred to as *picking time*. These devices automatically record detailed timestamps for every event in the warehouse, with precision down to the second, including decimal fractions of a second.

Order picking is the most time-intensive warehouse operation, accounting for 55% to 65% of total warehouse expenses. It is analogous to UPMSP, where stations correspond to machines M, tasks correspond to jobs N, and picking times correspond to processing times p_{jk}, for $j \in N, k \in M$. Additionally, each task

Table EC.1 Summary of the Performance of Different Versions of the BPC Algorithm (First Part).

Group			BPC		No DSUB		No Buck. Graph		No Domi. Rule		No Stro. Bran.	
m	n	inst.	Opt	Time [s]	Opt	Time [s]	Opt	Time [s]	Opt	Time [s]	Opt	Time [s]
4	40	20	20	3.79	20	4.13	20	5.41	0	t1	20	3.84
4	60	20	20	43.94	20	51.62	20	102.95	0	t1	20	44.50
4	80	20	20	240.42	20	267.06	20	753.25	0	t1	20	278.05
8	60	20	20	7.47	20	8.26	20	10.04	0	t1	20	8.21
8	80	20	20	27.89	20	30.49	20	50.21	0	t1	20	29.89
8	100	20	20	96.01	20	106.05	20	209.31	0	t1	20	113.93
12	60	20	20	4.21	20	6.47	20	5.30	0	t1	20	4.98
12	80	20	20	12.66	20	13.71	20	19.78	0	t1	20	15.11
12	100	20	20	36.70	20	37.54	20	61.48	0	t1	20	38.75
16	80	20	20	9.63	20	21.85	20	10.95	0	t1	20	11.46
16	100	20	20	29.48	20	64.92	20	37.24	0	t1	20	31.65
16	200	20	18	564.29	17	595.53	17	1098.09	0	t1	17	598.64
20	80	20	20	6.88	20	11.66	20	9.03	0	t1	20	7.34
20	100	20	20	20.83	20	39.06	20	30.78	0	t1	20	22.02
20	200	20	20	334.81	19	348.03	19	514.62	0	t1	19	337.68
Total		**300**	**298**		**296**		**296**		**0**		**296**	

Table EC.2 Summary of the Performance of Different Versions of BPC Algorithm (Second Part).

Group			No Lm-SRC				No Heur. Pric.		No Vari. Fixi.		No Sche. Enum.	
m	n	inst.	Opt	Time [s]	BPC Time [s]	Inc. [%]	Opt	Time [s]	Opt	Time [s]	Opt	Time [s]
4	40	20	20	3.86	3.79	1.85	20	6.87	20	4.36	20	4.04
4	60	20	20	53.91	43.94	22.69	20	98.01	20	48.13	20	48.37
4	80	20	20	247.18	240.42	2.81	20	654.32	20	259.41	20	269.28
8	60	20	20	17.84	7.47	138.82	20	13.16	20	8.34	20	15.95
8	80	20	20	71.47	27.89	156.26	20	43.98	20	32.27	20	59.53
8	100	20	20	270.43	96.01	181.67	20	116.47	20	113.64	20	198.04
12	60	20	20	15.06	4.21	257.72	20	7.68	20	6.39	20	6.06
12	80	20	20	34.36	12.66	171.41	20	16.10	20	17.62	20	17.46
12	100	20	20	88.90	36.70	142.23	20	40.28	20	41.84	20	41.83
16	80	20	20	12.27	9.63	27.41	20	16.05	20	11.58	20	10.97
16	100	20	20	31.00	29.48	5.16	20	58.04	20	38.24	20	36.35
16	200	20	16	545.39	499.82	9.12	14	915.02	17	586.38	17	1130.10
20	80	20	20	7.12	6.88	3.49	20	11.75	20	8.08	20	7.92
20	100	20	20	29.79	20.83	43.01	20	33.28	20	24.42	20	22.81
20	200	20	20	473.25	334.81	41.35	17	438.24	20	363.97	20	619.24
Total		**300**	**296**				**291**		**297**		**297**	

j is assigned a priority level, represented by a weight w_j (e.g., JD.com Plus member orders have higher priority than regular orders). Picking times are the most critical data, as explained below:

- **Data Source**. The picking times represent forecasted average values for future time periods rather than historical records, generated by JD.com's predictive analytics department. This team develops and maintains advanced ensemble models tailored for operational forecasting within JD.com's fulfillment network. The prediction models incorporate a comprehensive feature set encompassing: (1) operational variables (demand patterns, shift configurations, resource availability), (2) product characteristics (category classifications, inventory dynamics), (3) worker-specific factors (experience levels, performance history), (4) environmental

conditions (temperature, peer effects), and (5) temporal patterns. This integrated methodology synthesizes diverse data sources to generate robust predictions that capture the complex operational dynamics of modern fulfillment environments.

- **Unrelated**. Picking times exhibit variability due to multiple factors, such as picker experience for specific order types and environmental working conditions. For a task, the time required may differ across stations, and a station may also show variations in picking times across tasks.
- **High-Precision**. The high-precision picking times (expressed as rational numbers within the interval $[1, 25]$ seconds, with three significant decimal places) reflect the sophistication of these predictive models. By employing multivariate operational analysis, these models generate accurate efficiency forecasts that enable dynamic resource allocation and capacity planning based on anticipated.