Highlights

Iterative beam search algorithms for the permutation flowshop

Luc Libralesso,Pablo Andres Focke,Aurélien Secardin,Vincent Jost

- First use of an iterative beam search for the permutation flowshop
- Simple yet efficient heuristic guidance strategies
- Bi-directional search strategy to minimize the makespan variant
- state-of-the-art results on large instances
- 101 new best-so-far solutions on VRF instances (makespan minimization)
- 51 new best-so-far solutions on Taillard instances (flowtime minimization)

Iterative beam search algorithms for the permutation flowshop

Luc Libralesso[a,*], Pablo Andres Focke[a], Aurélien Secardin[a] and Vincent Jost[a]

[a] *Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France*

ARTICLE INFO

Keywords:
Heuristics
Iterative beam search
Permutation flowshop
Makespan
Flowtime

ABSTRACT

We study an iterative beam search algorithm for the permutation flowshop (makespan and flowtime minimization). This algorithm combines branching strategies inspired by recent branch-and-bounds and a guidance strategy inspired by the LR heuristic. It obtains competitive results on large instances compared to the state-of-the-art algorithms, reports many new-best-so-far solutions on the VFR benchmark (makespan minimization) and the Taillard benchmark (flowtime minimization) without using any NEH-based branching or iterative-greedy strategy.
The source code is available at: `https://github.com/librallu/dogs-pfsp`.

1. Introduction

In the flowshop problem, one has to schedule jobs, where each job has to follow the same route of machines. The goal is to find a job order that minimizes some criteria. The Permutation FlowShop Problem (PFSP) is a common (and fundamental) variant that imposes the machines to process jobs in the same order (thus, a permutation of jobs is enough to describe a solution). The permutation flowshop has been one of the most studied problems in the literature [35, 31] and has been considered on various industrial applications [16, 42]. We may also note that the permutation flowshop is at the origin of multiple other variants, for instance, the blocking permutation flowshop [45], the multiobjective permutation flowshop [20], the distributed permutation flowshop [11], the no-idle permutation flowshop [32], the permutation flowshop with buffers [28] and many others. Regarding the criteria to minimize, in this paper, we study two of the most studied objectives: the makespan (minimizing the completion time of the last job on the last machine) and the flowtime (minimizing the sum of completion times of each job on the last machine). According to the scheduling notation introduced by Graham, Lawler, Lenstra, and Rinnooy Kan [13], the makespan criterion is denoted $F_m|prmu|Cmax$ and the flowtime criterion $F_m|prmu|\sum C_i$.

Consider the following example instance with $m = 3$ machines with $n = 4$ jobs (j_1, j_2, j_3, j_4) with the job processing time matrix P defined as follows where $P_{j,m}$ indicates the processing time of job j on machine m:

$$P = \begin{pmatrix} 3 & 2 & 1 & 3 \\ 3 & 4 & 3 & 1 \\ 2 & 1 & 3 & 2 \end{pmatrix}$$

One possible solution can be described in Figure 1. This solution has a makespan (completion time of the last job on the last machine) of 18 and a flowtime (sum of completion times on the last machine) of $8 + 11 + 16 + 18 = 53$.

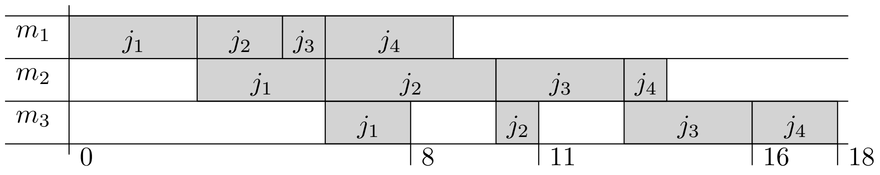


Figure 1: A solution for the example instance with a job order $\sigma = j_1, j_2, j_3, j_4$

luc.libralesso@grenoble-inp.fr (L. Libralesso); pablofocke@gmail.com (P.A. Focke); aurelien.secardin@icloud.com (A. Secardin); vincent.jost@grenoble-inp.fr (V. Jost)
ORCID(s):

Regarding resolution methods, the makespan minimization permutation flowshop problem has been massively studied over the last 50 years and a large number of numerical methods have been applied.

In 1983, Nawaz, Enscore, Ham proposed an insertion-based heuristic (later called NEH) [27]. This heuristic sorts jobs by some criterion (usually by a non-decreasing sum of processing times), then, it adds them one by one at the position that minimizes the objective function. The NEH, obtained, at the time, excellent results compared to other heuristics and can be used to perform greedy algorithms and perturbation-based algorithms as well. It has been largely considered as an essential component producing excellent solutions for large-scale permutation flowshop instances, and multiple methods have been built using it. One of the most famous ones is Taillard's acceleration [39]. It reduces the cost of inserting a job at all possible positions from $O(n^2.k)$ to $O(n.k)$. Considering these results, multiple works aim to improve the NEH heuristic [10, 26, 14, 4, 36, 44, 25] to quote a few.

The (meta-)heuristics state-of-the-art methods for the makespan minimization usually perform an iterated-greedy algorithm [38, 8]. Such algorithms start with a NEH heuristic to build an initial solution. Then, destroy a part of it and reconstruct it using again a NEH heuristic. To the best of our knowledge, the current state-of-the-art algorithms for the makespan minimization criterion are: the variable block insertion heuristic [15], the best-of-breed Iterated-Greedy [8], and, an automatically designed algorithm using the EMILI framework [29]. We may note that other algorithms exist to solve the makespan minimization. To quote a few, we can find some hybrid algorithms [46] (a combination of the NEH heuristic as a part of the initial population, a genetic algorithm, and simulated annealing to replace the mutation), memetic algorithms [17], an automatically designed local-search scheme [29].

The (meta-)heuristics methods for the flowtime minimization also involve the NEH heuristic, but some other constructive methods as well. For instance, the Liu and Reeve's method (LR) [24] performs a forward search (*i.e* appending jobs at the end of the partial schedule). It was later improved to reduce its complexity from $O(n^3m)$ to $O(n^2m)$, later called the FF algorithm [6]. Later, this scheme was integrated into a beam search algorithm (more on that later) that obtained state-of-the-art performance [7]. Recently, this beam search was integrated within a biased random-key genetic algorithm as a warm-start procedure [1]. In parallel, the authors of the EMILI framework also proposed an efficient algorithm for the flowtime minimization. These are, to the best of our knowledge, the state-of-the-art methods for the flowtime minimization alongside the algorithms proposed in [30].

Regarding exact-methods, a recent branch-and-bound [12] brought light on a bi-directional branching (*i.e* constructing the candidate solution from the beginning and the end at the same time) combined with a simple yet efficient bounding scheme to solve the makespan minimization criterion. The resulting branch-and-bound obtained excellent performance and was even able to solve to optimality almost all large VFR instances with 20 machines.

Moreover, recently, an iterative beam search has been proposed and, successfully applied to various combinatorial optimization problems as guillotine 2D packing problems [22, 9], the sequential ordering problem [21] and the longest common subsequence problem [23]. This iterative beam search scheme, at the beginning of the search, behaves as a greedy algorithm and then, more and more as a branch-and-bound algorithm as time goes (it performs a series of beam search iterations with a geometric growth). It naturally combines search-space reductions from branch-and-bounds and guidance strategies from classical (meta-)heuristics. Considering the success of recent branch-and-bound branching schemes and the performance of greedy-like algorithms to solve the permutation flowshop, it would be a natural idea to combine them. However, to the best of our knowledge, it has not been studied before. This paper aims to fill this gap. For the makespan criterion, we implemented a bi-directional branching scheme and combined it with a variant of the LR [24] guidance strategy and use an iterative beam-search algorithm to perform the search. We report competitive results compared to the state-of-the-art algorithms and find new best-known solutions on many large VFR instances (we improve the best-known solution for almost all instances with 500 jobs or more and 40 machines or more). Note that these results are interesting and new as almost all the efficient algorithms in the literature are based on the NEH heuristic or the iterated greedy algorithm. This is not the case for our algorithm as it is based on a variant of the LR heuristic and an exact-method branching scheme (bi-directional branching).

Regarding the flowtime criterion, the bi-directional branching cannot be directly applied (the bounding procedure is less efficient than for the makespan criterion). However, we show that an iterative beam search with a simple forward search (modified LR algorithm) is efficient, outperforms the current state-of-the-art algorithms, and, reports new best-solutions for the Taillard's benchmark (almost all solutions for instances with 100 jobs or more were improved).

This paper is structured as follows: Section 2 presents the iterative beam search strategy. Section 3 presents the branching schemes we implement (the forward and bi-directional search). Section 4 present the guides we implement (the bound guide, the idle-time guide, and mixes between these two first guides) and Section 5 presents the results obtained by running all variants described in this paper, showing that an iterative beam search combined with a simple variant of the LR heuristic can outperform the state-of-the-art.

2. The search strategy: Iterative beam search

Beam Search is a tree search algorithm that uses a parameter called the beam size (D). Beam Search behaves like a truncated *Breadth First Search (BrFS)*. It only considers the best D nodes on a given level. The other nodes are discarded. Usually, we use the bound of a node to choose the most promising nodes. It generalizes both a greedy algorithm (if $D = 1$) and a BrFS (if $D = \infty$). Figure 2 presents an example of beam search execution with a beam width $D = 3$.

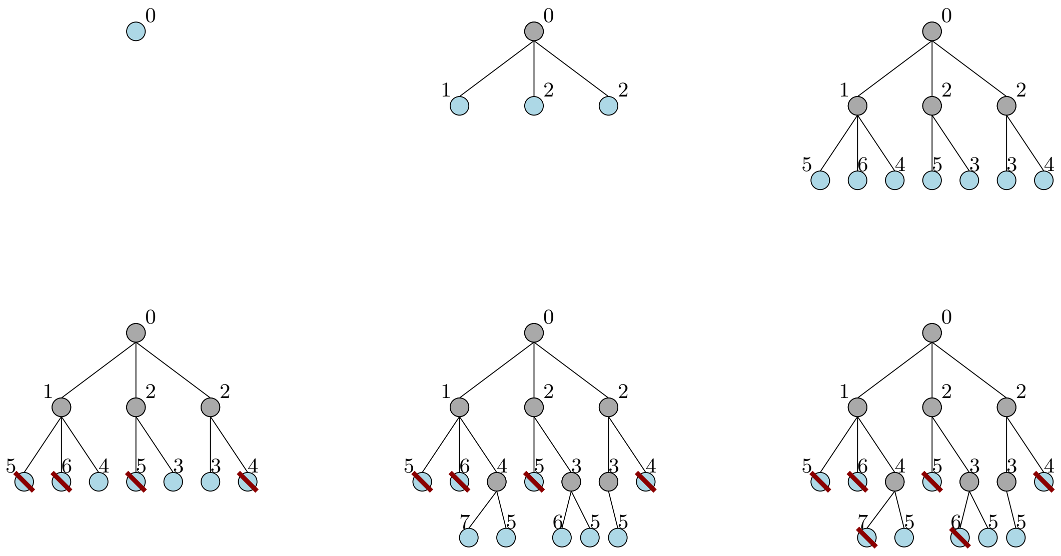


Figure 2: Beam Search Iterations with a beam width $D = 3$

Beam Search was originally proposed in [34] and used in speech recognition. It is an incomplete (*i.e.* performing a partial tree exploration and can miss optimal solutions) tree search parametrized by the beam width D. Thus, it is not an anytime algorithm. The parameter D allows controlling the quality of the solutions and the execution time. The larger D is, the longer it will take to reach feasible solutions, and the better these solutions will be.

Recently, a variant of beam search, called iterative beam search, was proposed and obtained state-of-the-art results on various combinatorial optimization problems [21, 22, 23, 9]. Iterative beam search performs a series of restarting beam search with geometrically increasing beam size until the time limit is reached. Algorithm 2.1 shows the pseudo-code of an iterative beam search. The algorithm runs multiple beam-searches starting with $D = 1$ (line 1) and increases the beam size (line 8) geometrically. Each run explores the tree with the given parameter D. In the pseudo-code, we increase geometrically the beam size by 2. This parameter can be tuned, however, we did not notice a significant variation in the performance while adjusting this parameter. This parameter (that can be a real number) should be strictly larger than 1 (for the beam to expand) and should not be too large, say less than 3 or 5 (otherwise, the beam grows too fast and when the time limit is reached, most of the computational time was possibly wasted in the last incomplete beam, without providing any solution).

The Section 3 presents the branching schemes used to generate children (Algorithm 2.1, line 5) and the Section 4 presents ways to identify the best nodes (Algorithm 2.1, line 6).

Algorithm 2.1: Iterative Beam Search algorithm

```
Input: root node
D ← 1
while stopping criterion not met do
    Candidates ← {root}
    while Candidates ≠ ∅ do
        nextLevel ← $\bigcup_{n \in \text{Candidates}}$ children($n$)
        Candidates ← best $D$ nodes among nextLevel
    end
    $D \leftarrow D \times 2$
end
```

3. Branching schemes

We present in this section the two branching schemes we use (*i.e.* the search tree structure): the forward search (*i.e* constructing the solution from the beginning) and the bi-directional search (*i.e.* constructing the solution from the beginning and the end).

3.1. Forward branching

The forward branching assigns jobs at the first free position in the partial sequences (it constructs the solutions from the beginning). The root corresponds to a situation where the candidate solution contains no job (*i.e.* $c.\text{STARTING} = \emptyset$). Each of the search-tree nodes correspond to the first jobs in the resulting solution. Children of a given node correspond to a possible insertion of each job that is not scheduled yet at the end of the schedule. Each node stores information about the partial candidate solution (jobs already added), the release time of each machine, and the partial makespan (resp. flowtime). A candidate solution (or node) c is considered as "goal" or "feasible" if all jobs are inserted (*i.e.* $c.\text{STARTING} = J$) and contains the following information:

- STARTING: vector of jobs inserted that lead to the candidate c (first jobs of the sequence we want to generate).
- FRONTSTARTING: vector of times where machines are first available after appending STARTING jobs.

Before presenting the forward children-generation, we present how to insert a job $j \in J$ in a candidate solution c (Algorithm 3.1). This insertion can be done in $O(m)$ where m is the number of machines.

Algorithm 3.1: Forward search: insertion of job j in candidate solution c (INSERTFORWARD(c, j))

```
Input: candidate solution (or node) $c$
Input: job to be inserted $j \in J$
$c.\text{FRONTSTARTING}_1 \leftarrow c.\text{FRONTSTARTING}_1 + P_{j,1}$
for $i \in \{2, \dots m\}$ do
    if $c.\text{FRONTSTARTING}_{i-1} > c.\text{FRONTSTARTING}_i$ then
        /* there is some idle time on machine $i$ */
        idle ← $c.\text{FRONTSTARTING}_{i-1} - c.\text{FRONTSTARTING}_i$
        $c.\text{FRONTSTARTING}_i \leftarrow c.\text{FRONTSTARTING}_{i-1} + P_{j,i}$
    else
        /* no idle time on machine $i$ */
        $c.\text{FRONTSTARTING}_i \leftarrow c.\text{FRONTSTARTING}_i + P_{j,i}$
    end
end
$c.\text{STARTING} \leftarrow c.\text{STARTING} \cup \{j\}$
```

Algorithm 3.2 presents the forward branching pseudo-code (how to generate all children of a candidate solution c).

Algorithm 3.2: Forward search children generation from a candidate solution c (CHILDREN(c))

```
Input: candidate solution (or node) $c$
1 children ← $\emptyset$
2 for $j \in$ unscheduled jobs do
3     children ← children ∪ INSERTFORWARD(Copy($c$), $j$)
4 end
5 return children
```

3.2. Bi-directional branching

To the best of our knowledge, bi-directional branching was first introduced in 1980 [33]. The bi-directional search appends jobs at the beginning and the end of the candidate solution. It aims to exploit the property of the inverse problem (job order inversed and machine order inversed). Since then, the efficiency of this scheme has been largely recognized to solve the makespan minimization optimally [2, 18, 19, 5, 3, 37]. Recently, a parallel branch-and-bound was successfully used to solve the makespan minimization criterion [12] using this bi-directional scheme. Multiple ways to decide if the algorithm performs a forward or backward insertion were studied (for instance alternating between a forward insertion and backward insertion). This study found out that the best way is selecting the insertion type that has the less remaining children after the bounding pruning step. Ties are broken by selecting the type of insertion that maximizes the sum of the lower bounds as large lower bounds are usually a more precise estimation.

A candidate solution (or node) c is considered as "goal" or "feasible" if all jobs are inserted (*i.e.* c.STARTING ∪ c.FINISHING $= J$) and contains the following information:

- STARTING: vector of jobs inserted at the beginning of the partial permutation that lead to the candidate c (first jobs of the sequence we want to generate).
- FRONTSTARTING: vector of times where machines are first available after appending STARTING jobs.
- FINISHING: (inverted) vector of jobs inserted at the end of the partial permutation that lead to the candidate c (last jobs of the sequence we want to generate).
- FRONTFINISHING: vector of times where machines are no more available after appending STARTING jobs.

Algorithm 3.3 presents the bi-directional branching pseudo-code. We use INSERTFORWARD (Algorithm 3.1) to insert a job within the STARTING vector and INSERTBACKWARD that inserts a job within the FINISHING vector. This procedure is almost similar to INSERTFORWARD but iterates over machines in an inverted order ($m \rightarrow 2$ instead of $2 \rightarrow m$). It generates children of both the forward and backward search (lines 1-6), prunes nodes that are dominated by the best-known solution (or upper-bound, lines 7-8). Then, it chooses the scheme that has fewer children (thus, usually a smaller search-space) and breaks ties by selecting the scheme having the more precise lower bounds (sum of lower bounds).

4. Guides

In the previous section, we discussed the branching rules that define a search tree. As such trees are usually large, a way to tell which node is apriori more desirable is needed. In branch-and-bounds, this mechanism is called "bound" and also constitutes an optimistic estimate of the best solution that can be achieved in a given sub-tree. In constructive meta-heuristics, the guidance strategy is usually not an optimistic estimate which often allows finding better solutions (for instance the LR [24] greedy guidance strategy). In this section, we present several guidance strategies for both the makespan and flowtime criteria.

4.1. Bound

We define the bound guidance strategy for the forward search and makespan minimization as follows. It measures the first time the last machine (machine m) is available and assumes that each remaining job can be scheduled without any idle time.

Algorithm 3.3: Bi-directional search children generation from a candidate solution c (CHILDREN(c))

```
Input: candidate solution (or node) $c$
1  F ← ∅                    /* F correspond to the children obtained by forward search */
2  B ← ∅                    /* B correspond to the children obtained by backward search */
3  for $j \in$ unscheduled jobs do
4      F ← F ∪ InsertForward(Copy($c$), $j$)
5      B ← B ∪ InsertBackward(Copy($c$), $j$)
6  end
7  F ← $\{c | c \in F$ if Bound($c$) < best known solution$\}$    /* removing forward nodes dominated by the UB */
8  B ← $\{c | c \in B$ if Bound($c$) < best known solution$\}$    /* removing backward nodes dominated by the UB */
9  if $|F| < |B| \lor (|F| = |B| \land \sum_{c \in F}$ Bound($c$) $> \sum_{c \in B}$ Bound($c$)) then
10     return F /* chosing the forward search */
11 else
12     return B /* chosing the backward search */
13 end
```

$$\mathrm{F}g_{\text{bound}} = Cmax_{f,m} + R_m$$

The bound guidance strategy for the bi-directional search and makespan minimization is defined as follows. It generalizes the bound for the forward search by also taking into account the backward front. We may note that the bi-directional branching allows computing a better bound as all machines are relevant for this bound (compared to the forward branching bound in which only the last machine is used to compute a bound).

$$\mathrm{FB}g_{\text{bound}} = \max_{i \in M}(Cmax_{f,i} + R_i + Cmax_{b,i})$$

The flowtime bound is defined as the sum of end times for each job scheduled in the forward search. Each time a job is added to the candidate solution, the flowtime value is modified.

4.2. idle time

The bound guide is an effective guidance strategy, but is known to be imprecise at the beginning of the search (*i.e.* the first levels of the search tree). Another guide that is usually considered as a part of effective greedy strategies (for instance the LR heuristic) is to use the idle time of the partial solution. Usually, a solution with a small idle time reaches good performance on both the makespan or flowtime criteria.

The idle time can be defined as follows:

$$\mathrm{FB}g_{\text{idle}} = \sum_{i \in M} I_{f,i} + I_{b,i}$$

4.3. bound and idle time

As it is noted in many works [24, 7], another interesting guidance strategy is to combine both guidance strategies discussed earlier (*i.e.* the bound and idle time guides). Indeed, while the bound guide is usually ineffective to guide the search close to the root, it is very precise close to feasible solutions. Inversely, the idle time is an efficient guide close to the root but relatively inefficient close to feasible solutions. We study the *bound and idle time guide* that linearly reduces the contribution of the idle time to favor the bound depending on the completion level of the candidate solution.

The bound and idle time guide can be defined as follows, where C is a value used to make the idle time and bound comparable:

$$g_{\text{alpha}} = \alpha \cdot g_{\text{bound}} + (1 - \alpha) \cdot C \cdot g_{\text{idle}}$$

where α corresponds to the proportion of jobs added (*i.e.* 0 if no jobs are added, 1 if all jobs are added). It is defined as follows: $\alpha = \frac{|F|+|B|}{|J|}$ for the bi-directional branching or $\alpha = \frac{|F|}{|J|}$ for the forward branching.

4.4. bound and weighted idle time

Another useful remark found in greedy algorithms for the permutation flowshop problem [24] is to add additional weight to the idle time produced by the first machines at the beginning of the search (as it will have a greater impact on the objective function than the others). However, the LR heuristic cannot be directly applied in a general tree search context. Indeed, it is sometimes noted [7] that algorithms like the beam search usually compare nodes from different parents, thus, it is needed to adapt the LR heuristic guidance that only compares nodes with the same parent. We propose two different simple yet efficient ways to implement similar ideas. The search is guided by a combination of a weighted idle time and by the bounding procedure.

The first guide, used for the forward search for the flowtime minimization is defined as follows, where I_w is the weighted idle time and $C = m.\frac{\sum_{i \in M} I_i}{2}$:

$$g_{\text{walpha}} = \alpha \ . \ g_{\text{bound}} + (1-\alpha) \ . \ (I_w + C)$$

At each time we add a job j to the end of the partial solution, we increase the weighted idle times as follows where v is the idle time added by the job j in machine i:

$$I_w = I_w + v \ . \ (\alpha \ . \ (m-i) + 1)$$

For the bi-directional branching, we present a new guidance strategy that considers the sum of idle time percentage for each front. Doing this, it allows making idle time on the first machines more important to the forward search and the idle time on the last machines more important to the backward search. The bound and weighted idle time guide for the bi-directional search is defined as follows:

$$g_{\text{wfrontalpha}} = (1-\alpha).g_{\text{bound}}.\left(\sum_{i \in M} \frac{I_{f,i}}{Cmax_{f,i}} + \frac{I_{b,i}}{Cmax_{b,i}}\right) + \alpha.g_{\text{bound}}$$

Notice that, during the bi-directional search, if only one direction is used (all jobs are inserted in the forward part (resp. backward part)), $g_{\text{wfrontalpha}}$ is not defined. We choose to consider that $g_{\text{wfrontalpha}} = \infty$ in this case. Indeed, using both fronts allows better bounds and guides, thus nodes using only one front should be not chosen over nodes that use both.

4.5. bound and gap

While solving some instances using a bi-directional branch-and-bound, we may notice that sometimes, the bound is very tight (thus is also a good guide). We propose a new guide that uses the gap between the best soulution found and the node bound ($\frac{UB-LB}{UB}$). If the gap is small (close to 0) the bound will be used more as a guide. If the gap is large, the idle time will be more considered. The "gap" guide is defined as follows:

$$g_{\text{gap}} = \frac{UB}{UB-LB}.g_{\text{bound}} + \frac{UB-LB}{UB}.\left(\sum_{i \in M} \frac{I_{f,i}}{Cmax_{f,i}} + \frac{I_{b,i}}{Cmax_{b,i}}\right)$$

Similarly to $g_{\text{wfrontalpha}}$, $g_{\text{gap}} = \infty$ if only one direction has been taken by the node.

5. Numerical results

In this section, we perform various experiments to evaluate the efficiency of the algorithms discussed in the previous sections. In Subsection 5.1, we present numerical results obtained in the makespan minimization version and Subsection 5.2, the results obtained in the flowtime minimization version. All algorithms have been implemented in rust and executed on an AMD Ryzen 5 3600 CPU @3.6GHz with 32GB RAM. As the CPU has multiple physical cores, we ran 4 tests in parallel to obtain results faster. For both objectives, we study the ARPD (Average Relative Percentage Deviation), defined as follows:

$$ARPD_{Ia} = \sum_{i \in I} \frac{M_{ai} - M_i^*}{M_i^*} \cdot \frac{100}{|I|}$$

where I is a set of instances with similar characteristics, M_{ai} corresponds to the objective obtained by algorithm a on instance i. And M_i^* the reference solution objective for instance i. The ARPD describes the performance of a given algorithm on a given instance type. For the makespan minimization (taillard benchmark), we used the best upper-bounds provided on Taillard's website[1]. For the makespan minimization (VFR benchmark, we used the best-results provided by [29][2]). For the flowtime minimization, we used the best solutions reported in [30].

For each instance and each criterion, we ran our algorithms for $n.m.45$ milliseconds where n is the number of jobs and m the number of machines as it is usually done in the literature. We evaluate our algorithms on the famous Taillard benchmark [40] (makespan and flowtime minimization) and on the famous VFR benchmark [43]. The first consists of sets of 10 instances with a job number $n \in \{20, 50, 100, 200, 500\}$ and machine number $m \in \{5, 10, 20\}$. The later consists of sets of 10 instances with a job number $n \in \{100, 200 \ldots 800\}$, a machine number $m \in \{20, 40, 60\}$. For each variant, we compare our algorithms with state-of-the-art algorithms.

5.1. Makespan minimization

5.1.1. Iterative beam search performance comparison

In Sections 3,4, we presented multiple variants of the Iterative beam search (forward and bi-directional search, 5 different guides). Figure 3 presents a performance comparison of the different iterative beam search algorithms we proposed.

Discussions: Regarding the forward branching procedures, we observe a significant improvement by including the idle time in the guide and obtain the best results by including a weighted idle time within the guide (similarly to the principles presented in the LR heuristic [24]). Indeed, ARPD ranges from 17% to 25% for the bound guide, and goes down between 1% to 5% for the idle and gap guides on the VFR instances. We note that the *wfrontalpha* and *gap* guides do not contribute much to the algorithm performance, and, surprisingly, simple guides (*idle, alpha*) perform better.

Regarding the bi-directional branching procedures, we observe that the bound guide performs well in most cases, from 0.16% to 8% ARPD. This can be explained as the bound gets tighter when the number of machines is low. Using the idle time in the guide (idle time only or idle time combined with the bound) decreases the performance of the algorithm (performances ranging from 2% to 17%). It seems to indicate that the idle time is a less efficient guide than the bound for this branching strategy. Finally, using the "front-weighted" idle time proves to be a significant bonus and largely improves the quality of the solutions, from −1.25% to 3% ARPD. The *gap* guide also allows improving the results obtained by the *bound* guide. These results show that the bi-directional search with the *wfrontalpha* guide performs well on most instances (especially those with a high number of machines) and the *gap* guide performs well on instances with fewer machines. Thus, we use these algorithms to compare with the state-of-the-art algorithms.

5.1.2. Comparison with the state-of-the-art algorithms

The best performing algorithms in the literature are: The Variable Block Insertion Heuristic (VBIH) [15], the Best-of-Breed Iterated Greedy algorithm (IGbob) [8], and, the Iterated Greedy designed using the EMILI framework (IGirms) [29]. Figure 4 compares the performance of our algorithms with the VBIH algorithm. VBIH results are obtained from the supplementary materials of [15]. CPU times are regularized to make a fair comparison. We do not include results on the Taillard dataset as the authors of VBIH only compared it using the VFR benchmark. Figure 5 compares the performance of our algorithms with the IGirms algorithm. IGirms results are obtained from the supplementary materials of [29][3]. IGirms authors provide their ARPD values but not the solutions obtained for each instance (thus, we cannot apply the Wilcoxon signed-rank test). Figure 6 compares the performance of our algorithms compared

[1]http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt
[2]http://iridia.ulb.ac.be/supp/IridiaSupp2018-002/
[3]http://iridia.ulb.ac.be/supp/IridiaSupp2018-002/

	Forward search					bi-directional search				
instance sets	bound	idle	alpha	wfrontalpha	gap	bound	idle	alpha	wfrontalpha	gap
TAI20_5	3.00	1.95	1.43	7.88	7.97	0.00	0.00	0.00	0.00	0.00
TAI20_10	6.28	0.89	0.92	11.17	11.27	0.03	0.36	0.36	0.00	0.03
TAI20_20	5.50	0.88	1.06	8.95	9.15	0.75	2.23	2.22	0.34	0.71
TAI50_5	6.39	0.89	1.07	9.01	9.13	0.00	0.00	0.00	0.11	0.00
TAI50_10	13.28	3.53	4.62	18.94	19.09	0.11	4.78	4.66	1.02	0.11
TAI50_20	14.58	3.04	3.39	20.49	20.29	2.86	7.55	7.23	**0.55**	2.91
TAI100_5	7.45	0.27	0.26	9.09	9.24	0.00	0.11	0.11	0.11	0.00
TAI100_10	14.20	1.52	1.46	15.01	15.21	0.00	2.55	2.32	1.18	0.00
TAI100_20	18.78	3.80	4.06	20.20	20.41	2.41	8.89	8.42	1.65	1.53
TAI200_10	13.66	1.27	1.36	12.31	12.37	0.00	1.49	1.48	2.24	1.05
TAI200_20	21.42	2.43	2.96	18.49	18.56	1.61	8.09	7.62	2.00	0.93
TAI500_20	20.29	1.77	1.99	14.24	14.20	0.65	4.63	5.30	1.49	**0.18**
VFR100_20	20.25	3.28	3.39	21.58	21.41	2.38	10.87	10.08	**0.52**	2.21
VFR100_40	17.68	4.81	5.48	19.83	19.73	5.66	10.00	9.61	**1.65**	5.57
VFR100_60	16.37	5.34	6.35	17.60	17.59	6.68	9.96	9.75	**3.02**	6.58
VFR200_20	21.12	2.65	2.78	19.54	19.58	1.37	11.13	10.12	1.12	0.77
VFR200_40	21.67	4.88	5.44	19.64	19.93	5.65	15.40	14.73	**0.07**	5.46
VFR200_60	20.09	5.65	5.96	18.83	18.75	8.20	16.03	15.53	**1.82**	7.84
VFR300_20	20.76	2.00	2.06	17.30	17.24	0.73	7.87	7.53	1.23	**0.11**
VFR300_40	23.29	4.63	5.04	18.69	18.75	6.08	17.13	16.91	**-0.48**	5.75
VFR300_60	20.47	5.66	6.00	18.05	18.02	7.86	12.66	11.99	**1.08**	7.96
VFR400_20	21.48	1.69	1.88	15.20	15.24	0.52	4.40	4.05	0.93	**0.04**
VFR400_40	23.40	4.15	4.60	17.92	17.88	5.71	16.17	16.28	**-0.85**	5.81
VFR400_60	21.12	5.73	6.11	17.30	17.25	7.91	17.39	17.49	**0.07**	8.37
VFR500_20	19.94	1.28	1.27	14.45	14.59	0.41	3.27	3.23	0.94	**-0.01**
VFR500_40	22.75	3.58	4.02	17.37	17.35	5.14	13.93	13.69	**-0.66**	5.21
VFR500_60	21.54	6.11	6.31	16.50	16.48	8.50	17.08	16.80	**-0.61**	7.91
VFR600_20	19.64	1.18	1.07	13.49	13.55	0.29	3.54	3.41	0.92	**-0.08**
VFR600_40	23.69	3.73	3.91	16.37	16.38	5.77	16.18	16.86	**-0.50**	5.41
VFR600_60	21.60	5.91	6.11	16.01	15.99	8.22	13.07	12.35	**-0.80**	7.58
VFR700_20	19.39	0.96	1.12	12.54	12.65	0.23	2.50	2.33	1.01	**-0.11**
VFR700_40	23.49	3.63	3.49	15.93	16.01	4.73	12.91	12.35	**-0.37**	4.33
VFR700_60	22.42	5.91	6.15	15.87	15.78	8.35	16.18	15.71	**-1.06**	7.95
VFR800_20	20.28	0.99	0.96	11.71	11.76	0.24	2.16	2.14	0.83	**-0.07**
VFR800_40	22.84	3.60	3.78	15.32	15.35	4.11	14.79	14.77	**-0.38**	3.99
VFR800_60	22.45	6.14	6.47	15.50	15.52	8.16	16.04	15.81	**-1.11**	7.94

Figure 3: Average Relative Percentage Deviation (ARPD) of all the presented algorithms on the Taillard and VFR instances for the makespan minimization version. Bold values indicate that the algorithm obtained significantly better results then the others according to the Wilcoxon signed-rank test with a 95% confidence interval.

to the IGbob algorithm [8]. As the authors provide their source-code, we executed their algorithm on our machine. Figure 10 presents Pareto diagrams showing the time/performance tradeoff of our algorithms and the state-of-the-art algorithms for 2 of the largest instance families (VFR800_20, VFR800_60).

discussions: From Tables 4,5,6, we remark that the *wfrontalpha iterative beam search* perform significantly better on large instances (more than 500 jobs and 40 machines). It often reports negative ARPD (meaning that it was able to consistently report new-best-known solutions compared to IGirms), even on short computation times. It also has to be noted that on the Pareto diagrams 10, the iterative beam searches find better solutions in shorter computation times on large instances than all the reported state-of-the-art results. Moreover, it can report new-best-known solutions on large classes of instances in 200 seconds for the VFR800_20 instances and 80 seconds for the VFR800_60 instances.

	$n.m.30/2$ CPU-regularized ms			$n.m.60/2$ CPU-regularized ms			$n.m.90/2$ CPU-regularized ms		
instance set	VBIH	IBS wfrontalpha	IBS gap	VBIH	IBS wfrontalpha	IBS gap	VBIH	IBS wfrontalpha	IBS gap
VFR100_20	**0.27**	0.73	2.98	**0.23**	0.65	2.50	**0.04**	0.65	2.21
VFR100_40	**0.52**	2.23	6.65	**0.46**	1.91	6.05	**0.26**	1.74	5.57
VFR100_60	**0.63**	3.80	7.44	**0.57**	3.52	6.94	**0.41**	3.02	6.58
VFR200_20	**0.22**	1.15	1.40	**0.20**	1.13	1.18	**0.09**	1.13	0.80
VFR200_40	0.56	0.59	6.46	0.52	0.35	5.84	0.24	**0.07**	5.71
VFR200_60	**0.61**	2.78	9.21	**0.58**	2.22	8.29	**0.32**	1.82	7.84
VFR300_20	**0.20**	1.36	0.73	**0.16**	1.27	0.21	0.12	1.26	0.12
VFR300_40	0.53	**-0.09**	6.56	0.49	**-0.30**	5.92	0.32	**-0.48**	5.75
VFR300_60	**0.66**	1.48	8.31	**0.62**	1.08	7.96	**0.39**	1.08	7.96
VFR400_20	**0.15**	1.03	0.30	0.12	0.94	0.10	**0.07**	0.93	0.10
VFR400_40	0.47	**-0.63**	6.35	0.43	**-0.78**	6.07	0.27	**-0.85**	5.81
VFR400_60	0.58	0.67	8.85	0.54	0.36	8.78	0.32	**0.07**	8.37
VFR500_20	**0.12**	1.02	0.20	0.11	0.97	0.05	0.05	0.94	-0.01
VFR500_40	0.55	**-0.56**	5.46	0.49	**-0.63**	5.26	0.34	**-0.63**	5.26
VFR500_60	0.41	**-0.08**	8.90	0.37	**-0.51**	8.45	0.18	**-0.51**	8.45
VFR600_20	0.12	1.11	0.09	0.11	1.00	0.04	0.09	1.00	-0.00
VFR600_40	0.50	**-0.45**	5.44	0.37	**-0.47**	5.44	0.27	**-0.50**	5.41
VFR600_60	0.64	**-0.55**	7.77	0.50	**-0.69**	7.58	0.43	**-0.69**	7.58
VFR700_20	0.10	1.04	**-0.02**	0.06	1.02	**-0.09**	0.05	1.02	**-0.10**
VFR700_40	0.42	**-0.28**	4.75	0.28	**-0.32**	4.37	0.20	**-0.37**	4.33
VFR700_60	0.55	**-0.63**	8.38	0.41	**-0.89**	7.98	0.31	**-1.06**	7.95
VFR800_20	0.07	0.88	0.01	0.06	0.86	-0.02	0.05	0.83	-0.07
VFR800_40	0.32	**-0.26**	4.25	0.30	**-0.35**	4.15	0.22	**-0.38**	4.15
VFR800_60	0.41	**-0.77**	8.09	0.37	**-0.91**	7.94	0.29	**-1.11**	7.94

Figure 4: Comparison with VBIH. Bold values indicate that the algorithm obtained significantly better results then the others according to the Wilcoxon signed-rank test with a 95% confidence interval.

5.2. Flowtime minimization

5.2.1. Comparison with the state-of-the-art algorithms

The best performing algorithms in the literature are: IGA [30], ALGirtct [29], MRSILS(CBSH) [7] and Shake-LS [1]. Figure 8 compares our algorithms with ALGirtct and IGA with the results presented in [29]. Figure 9 compares our algorithms with MRSILS(CBSH). For both tables, the authors provide their ARPD values but not the solutions obtained for each instance (thus, we cannot apply the Wilcoxon signed-rank test). Figure 10 presents Pareto diagrams to evaluate our algorithms with state-of-the-art algorithms. Finally, the authors of Shake-LS report the best results obtained by their algorithm (30 independent runs of 1 hour). We do not directly compare our running times (that are much shorter), but we are still able to report many new-best-known solutions in Appendix B showing that our algorithm can compete with Shake-LS. All the algorithms presented above perform their experiments on the Taillard dataset [40].

discussions: We observe that both algorithms perform well for many instances and find new-best-known solutions. By contrast with the makespan minimization, both guidance strategies are comparable in terms of performance (the weighted idle time did not have a significant impact): sometimes g_{alpha} performs better than g_{walpha} and vice-versa. We may note that the main difference between our results and the beam search algorithms found in the literature [7] is that we use an iterative beam search that allows performing beam search with larger if the remaining time allows it. This result seems to indicate that the iterative beam search can be of interest to the community as it reports good results compared to other search strategies.

6. Conclusions & perspectives

In this paper, we present some iterative beam search algorithms applied to the permutation flowshop problem (makespan and flowtime minimization). These algorithms use branching strategies inspired by the LR heuristic (forward branching) and recent branch-and-bound schemes [12] (bi-directional branching). We compare several guidance

	$n.m.60/2$ CPU-regularized ms			$n.m.120/2$ CPU-regularized ms			$n.m.240/2$ CPU-regularized ms		
instance set	IGirms	IBS wfrontalpha	IBS gap	IGirms	IBS wfrontalpha	IBS gap	IGirms	IBS wfrontalpha	IBS gap
TAI20_5	0.03	0.00	0.00	0.02	0.00	0.00	0.01	0.00	0.00
TAI20_10	0.01	0.03	0.26	0.01	0.00	0.07	0.01	0.00	0.07
TAI20_20	0.01	0.56	1.09	0.01	0.51	0.88	0.01	0.34	0.72
TAI50_5	0.00	0.21	0.00	0.00	0.17	0.00	0.00	0.12	0.00
TAI50_10	0.30	1.52	0.15	0.28	1.22	0.13	0.25	1.02	0.11
TAI50_20	0.47	1.01	3.55	0.39	0.87	3.24	0.34	0.64	2.92
TAI100_5	0.00	0.27	0.00	0.00	0.20	0.00	0.00	0.11	0.00
TAI100_10	0.03	1.63	0.16	0.03	1.33	0.05	0.02	1.28	0.00
TAI100_20	0.62	2.19	2.32	0.52	2.06	2.12	0.44	1.78	1.53
TAI200_10	0.03	3.17	1.41	0.03	2.75	1.25	0.03	2.41	1.14
TAI200_20	0.66	2.06	1.54	0.57	2.04	1.14	0.51	2.00	1.04
TAI500_20	0.29	1.53	0.45	0.26	1.51	0.32	0.24	1.49	0.18
VFR100_20	0.57	0.83	3.33	0.42	0.73	2.74	0.29	0.65	2.21
VFR100_40	0.67	2.30	6.65	0.49	2.02	6.05	0.35	1.87	6.05
VFR100_60	0.64	3.80	7.44	0.48	3.52	6.94	0.34	3.02	6.58
VFR200_20	0.45	1.19	1.49	0.32	1.15	1.18	0.22	1.13	0.80
VFR200_40	0.79	0.73	6.53	0.52	0.48	6.15	0.30	0.35	5.84
VFR200_60	0.74	2.78	9.21	0.50	2.22	8.29	0.28	1.82	7.84
VFR300_20	0.35	1.41	0.86	0.24	1.36	0.59	0.17	1.26	0.12
VFR300_40	0.70	-0.09	6.56	0.48	-0.30	5.92	0.24	-0.48	5.75
VFR300_60	0.77	1.81	8.88	0.53	1.48	8.31	0.29	1.08	7.96
VFR400_20	0.21	1.11	0.33	0.16	1.02	0.23	0.12	0.93	0.10
VFR400_40	0.62	-0.63	6.35	0.42	-0.78	6.11	0.22	-0.85	6.07
VFR400_60	0.68	0.67	8.85	0.46	0.36	8.78	0.23	0.07	8.37
VFR500_20	0.17	1.02	0.22	0.13	0.97	0.06	0.09	0.94	-0.01
VFR500_40	0.54	-0.38	5.75	0.37	-0.56	5.46	0.20	-0.63	5.26
VFR500_60	0.61	0.29	9.10	0.41	-0.08	8.90	0.21	-0.51	8.45
VFR600_20	0.17	1.16	0.10	0.13	1.11	0.07	0.09	1.00	-0.00
VFR600_40	0.52	-0.37	5.44	0.34	-0.45	5.44	0.17	-0.50	5.41
VFR600_60	0.62	-0.28	7.88	0.42	-0.55	7.77	0.21	-0.69	7.58
VFR700_20	0.14	1.07	-0.01	0.10	1.04	-0.02	0.07	1.02	-0.10
VFR700_40	0.48	-0.28	4.75	0.31	-0.32	4.37	0.15	-0.37	4.33
VFR700_60	0.57	-0.63	8.38	0.37	-0.89	7.98	0.19	-1.06	7.95
VFR800_20	0.15	0.88	0.03	0.10	0.86	-0.01	0.07	0.83	-0.07
VFR800_40	0.46	-0.26	4.25	0.29	-0.35	4.25	0.14	-0.38	4.15
VFR800_60	0.51	-0.77	8.09	0.32	-0.91	7.94	0.15	-1.11	7.94

Figure 5: Comparison with IGirms

strategies (starting from the bound as commonly done in most branch-and-bounds) to more advanced ones (LR-inspired guidance). We show that the combination of all of these components obtains state-of-the-art performance. We report 101 new-best-so-far solutions for the permutation flowshop (makespan minimization) on the open instances of the VFR benchmark and 51 new-best-so-far solutions for the permutation flowshop (flowtime minimization) on the open instances of the Taillard benchmark. These algorithms compare, and sometimes perform better, than the algorithms based on the NEH branching scheme (which is usually considered as "the most efficient constructive heuristic for the problem" [8]) and the iterated greedy algorithm (again considered as "the most efficient approximate algorithm for the problem" [8]). We believe that the performance of the bi-directional branching combined to the iterative beam search highlighted in this paper could draw the interest of the community for these techniques as they are rather unexplored, although simple and efficient. Studying these techniques leads to a few other questions: we considered the iterative beam search and showed that it is competitive with classical meta-heuristics for the permutation flowshop. However, many other exist. For instance Iterative Memory Bounded A* [9, 22], Beam Stack Search [47], Anytime Column Search [41]. To the best of our knowledge, they have not been tested yet for the permutation flowshop. In this paper, we studied the makespan and flowtime minimization criteria and achieved competitive results. Many more flowshop

	$n.m.30/2$ ms			$n.m.60/2$ ms			$n.m.90/2$ ms		
instance set	IGbob	IBS wfrontalpha	IBS gap	IGbob	IBS wfrontalpha	IBS gap	IGbob	IBS wfrontalpha	IBS gap
TAI20_5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TAI20_10	0.00	0.03	0.11	0.00	0.00	0.07	0.00	0.00	0.03
TAI20_20	**0.00**	0.51	0.93	**0.00**	0.34	0.81	**0.00**	0.34	0.71
TAI50_5	0.00	0.17	0.00	0.00	0.12	0.00	0.00	0.11	0.00
TAI50_10	0.33	1.40	0.13	0.30	1.13	0.11	0.28	1.02	0.11
TAI50_20	**0.43**	0.87	3.24	**0.33**	0.64	2.92	**0.31**	0.55	2.91
TAI100_5	0.00	0.27	0.00	0.00	0.19	0.00	0.00	0.11	0.00
TAI100_10	0.03	1.37	0.05	0.02	1.30	0.01	0.02	1.18	0.00
TAI100_20	**0.60**	2.06	2.12	**0.52**	1.78	1.53	**0.50**	1.65	1.53
TAI200_10	0.03	2.75	1.25	0.03	2.41	1.14	0.03	2.24	1.05
TAI200_20	0.62	2.04	1.14	0.60	2.00	1.04	0.51	2.00	0.93
TAI500_20	0.26	1.52	0.32	0.24	1.50	0.28	0.23	1.49	0.18
VFR100_20	**0.53**	0.73	2.74	**0.38**	0.65	2.21	0.40	0.52	2.21
VFR100_40	**0.67**	2.23	6.65	**0.55**	1.91	6.05	**0.46**	1.65	5.57
VFR100_60	**0.75**	3.52	6.94	**0.52**	3.02	6.58	**0.46**	3.02	6.58
VFR200_20	**0.38**	1.15	1.18	0.33	1.13	0.80	0.22	1.12	0.77
VFR200_40	0.76	0.59	6.46	0.50	0.35	5.84	0.39	**0.07**	5.46
VFR200_60	**0.72**	2.22	8.29	**0.51**	2.06	7.84	**0.41**	1.82	7.84
VFR300_20	0.39	1.36	0.66	0.26	1.27	0.21	0.21	1.23	0.11
VFR300_40	0.64	**-0.30**	5.92	0.47	**-0.48**	5.75	0.33	**-0.48**	5.75
VFR300_60	**0.73**	1.48	8.31	**0.53**	1.08	7.96	**0.39**	1.08	7.96
VFR400_20	0.25	1.02	0.30	0.18	0.93	0.10	0.15	0.93	0.04
VFR400_40	0.52	**-0.63**	6.35	0.34	**-0.78**	6.07	0.21	**-0.85**	5.81
VFR400_60	0.58	0.36	8.78	0.37	0.19	8.65	0.28	0.07	8.37
VFR500_20	0.18	1.01	0.12	0.15	0.96	**-0.01**	0.11	0.94	-0.01
VFR500_40	0.54	**-0.56**	5.46	0.37	**-0.63**	5.26	0.25	**-0.66**	5.21
VFR500_60	0.41	**-0.08**	8.90	0.23	**-0.51**	8.45	0.12	**-0.61**	7.91
VFR600_20	0.14	1.11	0.08	0.11	1.00	0.01	0.08	0.92	**-0.08**
VFR600_40	0.35	**-0.45**	5.44	0.18	**-0.50**	5.41	0.09	**-0.50**	5.41
VFR600_60	0.46	**-0.55**	7.77	0.32	**-0.69**	7.58	0.19	**-0.80**	7.58
VFR700_20	0.11	1.04	**-0.02**	0.08	1.02	**-0.10**	0.07	1.01	**-0.11**
VFR700_40	0.34	**-0.32**	4.63	0.19	**-0.37**	4.37	0.12	**-0.37**	4.33
VFR700_60	0.42	**-0.89**	7.98	0.24	**-1.06**	7.95	0.13	**-1.06**	7.95
VFR800_20	0.09	0.86	-0.01	0.06	0.83	**-0.05**	0.05	0.83	-0.07
VFR800_40	0.31	**-0.26**	4.25	0.18	**-0.35**	4.15	0.09	**-0.38**	3.99
VFR800_60	0.37	**-0.91**	8.01	0.24	**-1.02**	7.94	0.15	**-1.11**	7.94

Figure 6: Comparison with IGbob. Bold values indicate that the algorithm obtained significantly better results then the others according to the Wilcoxon signed-rank test with a 95% confidence interval.

variants have been studied. For instance, the blocking flowshop, the distributed permutation flowshop and many others. It could be interesting to assess the performance of the LR-based beam search on these variants.

References

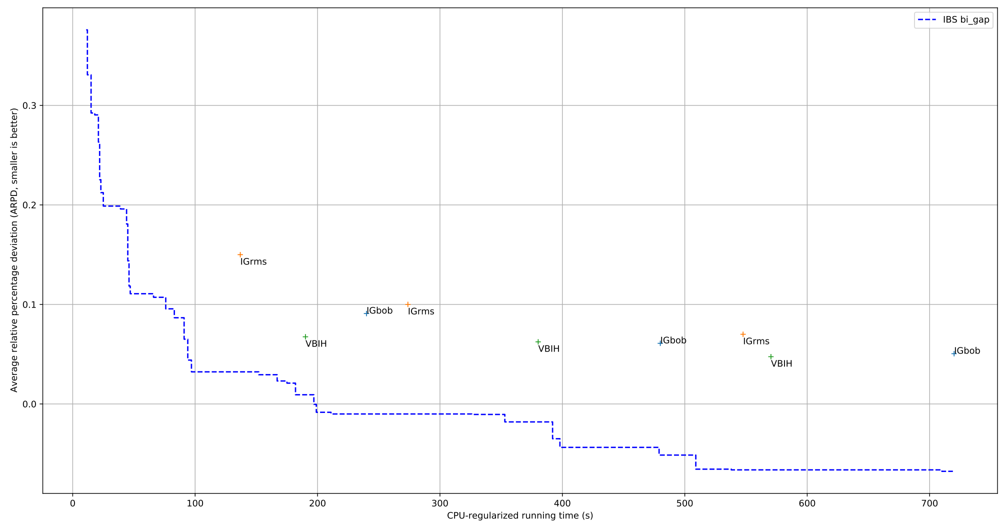
[1] Andrade, C.E., Silva, T., Pessoa, L.S., 2019. Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm. Expert Systems with Applications 128, 67–80.

[2] Carlier, J., Rebaï, I., 1996. Two branch and bound algorithms for the permutation flow shop problem. European Journal of Operational Research 90, 238–251.

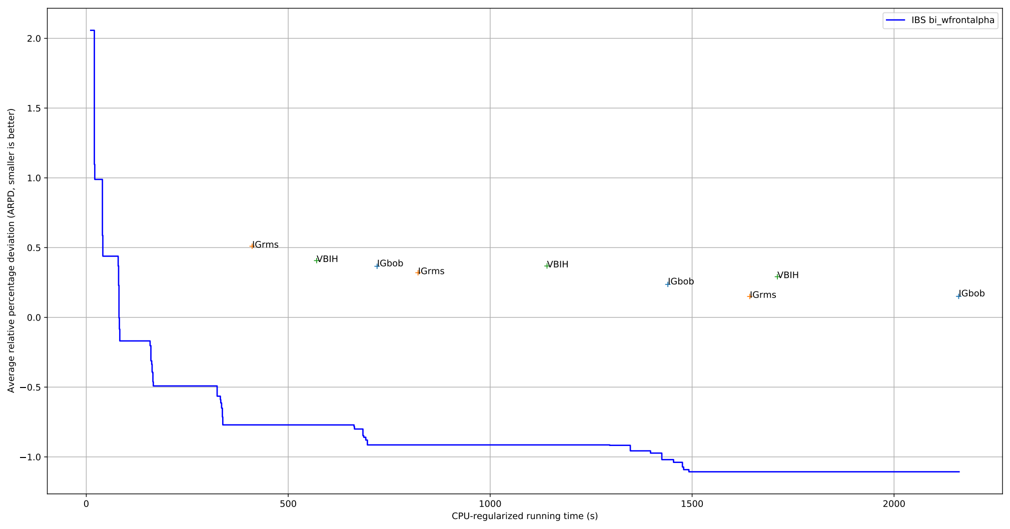
[3] Chakroun, I., Melab, N., Mezmaz, M., Tuyttens, D., 2013. Combining multi-core and gpu computing for solving combinatorial optimization problems. Journal of Parallel and Distributed Computing 73, 1563–1577.

[4] Dong, X., Huang, H., Chen, P., 2008. An improved neh-based heuristic for the permutation flowshop problem. Computers & Operations Research 35, 3962–3968.

[5] Drozdowski, M., Marciniak, P., Pawlak, G., Płaza, M., 2011. Grid branch-and-bound for permutation flowshop, in: International Conference on Parallel Processing and Applied Mathematics, Springer. pp. 21–30.



(a) Aggregated results for VFR800_20_X instances



(b) Aggregated results for VFR800_60_X instances

Figure 7: solution-quality/time pareto diagram comparing IBS algorithms with the state-of-the-art meta-heuristics on the largest VFR instances (makespan minimization).

[6] Fernandez-Viagas, V., Framinan, J.M., 2015. A new set of high-performing heuristics to minimise flowtime in permutation flowshops. Computers & Operations Research 53, 68–80.

[7] Fernandez-Viagas, V., Framinan, J.M., 2017. A beam-search-based constructive heuristic for the pfsp to minimise total flowtime. Computers & Operations Research 81, 167–177.

[8] Fernandez-Viagas, V., Framinan, J.M., 2019. A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. Computers & Operations Research 112, 104767.

[9] Fontan, F., Libralesso, L., 2020. Packingsolver: a tree search-based solver for two-dimensional two-and three-staged guillotine packing

	$n.m.60/2$ CPU-regularized ms				$n.m.120/2$ CPU-regularized ms				$n.m.240/2$ CPU-regularized ms			
instance set	ALGirtct	IGA	IBS alpha	IBS walpha	ALGirtct	IGA	IBS alpha	IBS walpha	ALGirtct	IGA	IBS alpha	IBS walpha
TAI20_5	0.00	0.15	0.01	0.24	0.00	0.15	0.00	0.13	0.00	0.15	0.00	0.01
TAI20_10	0.00	0.00	0.00	0.63	0.00	0.00	0.00	0.49	0.00	0.00	0.00	0.43
TAI20_20	0.00	0.00	0.00	0.97	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.91
TAI50_5	0.47	0.64	0.51	0.16	0.38	0.54	0.41	0.13	0.31	0.48	0.39	0.10
TAI50_10	0.51	1.10	0.54	0.96	0.41	1.04	0.53	0.94	0.35	0.99	0.46	0.86
TAI50_20	0.45	0.72	0.26	1.20	0.35	0.66	0.23	1.19	0.29	0.61	0.20	1.19
TAI100_5	0.99	1.17	0.01	-0.06	0.89	1.08	-0.04	-0.10	0.81	0.99	-0.06	-0.13
TAI100_10	1.03	1.49	0.19	0.22	0.90	1.37	0.15	0.16	0.79	1.29	0.07	0.02
TAI100_20	1.15	1.54	0.17	1.11	0.97	1.40	0.05	1.08	0.83	1.30	-0.02	0.91
TAI200_10	0.86	1.27	-0.59	-0.80	0.73	1.17	-0.68	-0.88	0.64	1.09	-0.73	-0.97
TAI200_20	0.70	1.09	-0.87	-0.61	0.53	0.92	-1.01	-0.76	0.39	0.80	-1.12	-0.80
TAI500_20	0.63	0.49	-1.69	-1.95	0.42	0.42	-1.89	-2.04	0.24	0.36	-1.98	-2.14

Figure 8: Comparison with ALGirtct and IGA for the flowtime minimization variant (Taillard benchmark)

instance set	MRSILS(CBSH)	IBS alpha	IBS walpha
TAI20_5	0.01	0.00	0.01
TAI20_10	0.00	0.00	0.49
TAI20_20	0.00	0.00	0.91
TAI50_5	0.28	0.41	0.13
TAI50_10	0.47	0.46	0.86
TAI50_20	0.63	0.20	1.19
TAI100_5	0.22	-0.04	-0.10
TAI100_10	0.27	0.07	0.02
TAI100_20	0.83	-0.02	0.91
TAI200_10	-0.71	-0.73	-0.97
TAI200_20	-0.83	-1.12	-0.80
TAI500_20	-1.90	-1.89	-2.04

Figure 9: Comparison with MRSILS(CBSH) with the same running times ($n.m.30$ CPU-regularized ms)

problems .

[10] Framinan, J.M., Leisten, R., 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega 31, 311–317.

[11] Gao, J., Chen, R., 2011. A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. International Journal of Computational Intelligence Systems 4, 497–508.

[12] Gmys, J., Mezmaz, M., Melab, N., Tuyttens, D., 2020. A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. European Journal of Operational Research .

[13] Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey, in: Annals of discrete mathematics. Elsevier. volume 5, pp. 287–326.

[14] Kalczynski, P.J., Kamburowski, J., 2008. An improved neh heuristic to minimize makespan in permutation flow shops. Computers & Operations Research 35, 3001–3008.

[15] Kizilay, D., Tasgetiren, M.F., Pan, Q.K., Gao, L., . A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion 12, 100. URL: https://www.mdpi.com/1999-4893/12/5/100, doi:10.3390/a12050100. number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[16] Krajewski, L.J., King, B.E., Ritzman, L.P., Wong, D.S., 1987. Kanban, mrp, and shaping the manufacturing environment. Management science 33, 39–57.

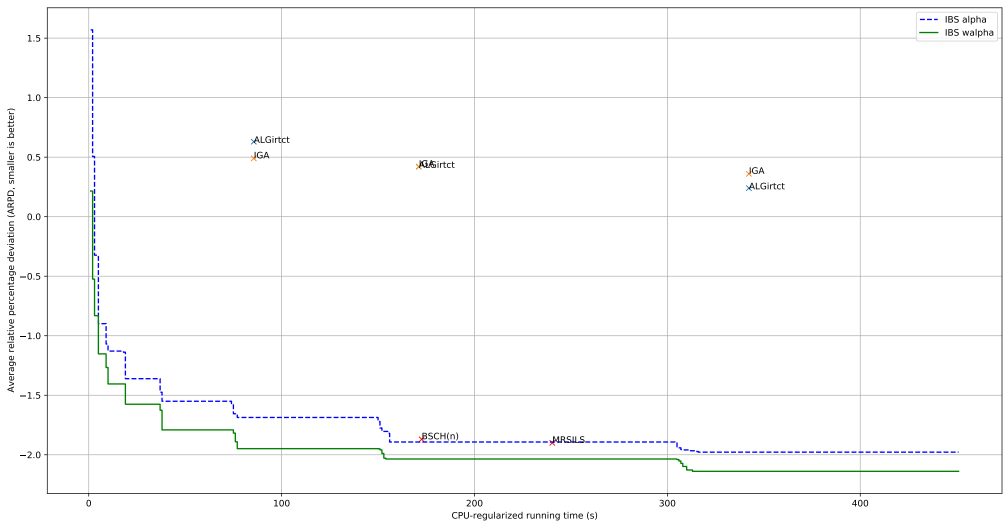
[17] Kurdi, M., 2020. A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. Applied Soft Computing , 106458.

[18] Ladhari, T., Haouari, M., 2005. A computational study of the permutation flow shop problem based on a tight lower bound. Computers & Operations Research 32, 1831–1847.

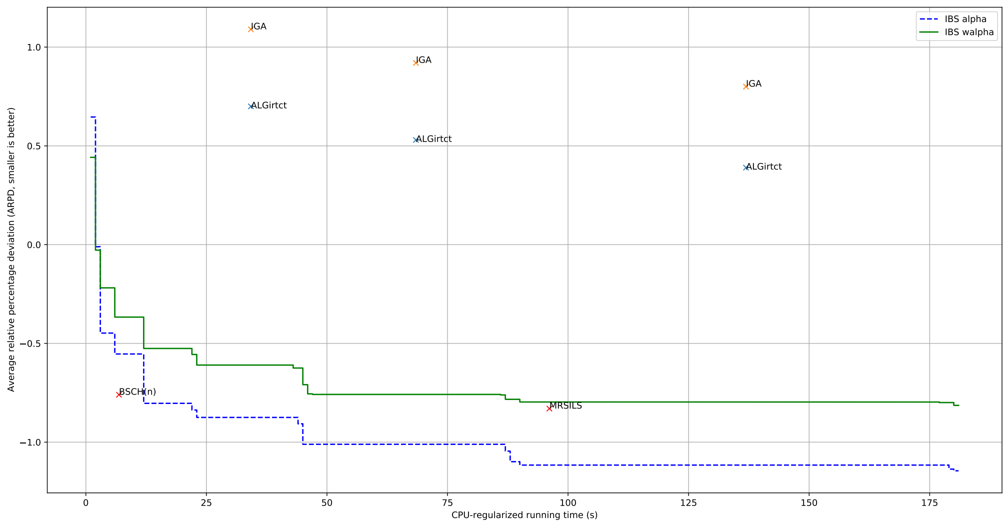
[19] Lemesre, J., Dhaenens, C., Talbi, E.G., 2007. An exact parallel method for a bi-objective permutation flowshop problem. European Journal of Operational Research 177, 1641–1655.

[20] Li, B.B., Wang, L., Liu, B., 2008. An effective pso-based hybrid algorithm for multiobjective permutation flow shop scheduling. IEEE transactions on systems, man, and cybernetics-part A: systems and humans 38, 818–831.

[21] Libralesso, L., Bouhassoun, A.M., Cambazard, H., Jost, V., 2019. Tree search algorithms for the sequential ordering problem. arXiv preprint arXiv:1911.12427 .



(a) Aggregated results for TAI500_20_X instances



(b) Aggregated results for TAI200_20_X instances

Figure 10: solution-quality/time pareto diagram comparing IBS algorithms with the state-of-the-art meta-heuristics on the largest Taillard instances (flowtime minimization).

[22] Libralesso, L., Fontan, F., 2020. An anytime tree search algorithm for the 2018 roadef/euro challenge glass cutting problem. arXiv preprint arXiv:2004.00963 .

[23] Libralesso, L., Secardin, A., Jost, V., 2020. Longest common subsequence: an algorithmic component analysis. URL: https://hal.archives-ouvertes.fr/hal-02895115. working paper or preprint.

[24] Liu, J., Reeves, C.R., 2001. Constructive and composite heuristic solutions to the $p//\sum c_i$ scheduling problem. European Journal of Operational Research 132, 439–452.

[25] Liu, W., Jin, Y., Price, M., 2017. A new improved neh heuristic for permutation flowshop scheduling problems. International Journal of

Production Economics 193, 21–30.
[26] Nagano, M., Moccellin, J., 2002. A high quality solution constructive heuristic for flow shop sequencing. Journal of the Operational Research Society 53, 1374–1379.
[27] Nawaz, M., Enscore Jr, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11, 91–95.
[28] Nowicki, E., 1999. The permutation flow shop with buffers: A tabu search approach. European Journal of Operational Research 116, 205–219.
[29] Pagnozzi, F., Stützle, T., 2019. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. European Journal of Operational Research 276, 409–421.
[30] Pan, Q.K., Ruiz, R., 2012. Local search methods for the flowshop scheduling problem with flowtime minimization. European Journal of Operational Research 222, 31–43.
[31] Pan, Q.K., Ruiz, R., 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Computers & Operations Research 40, 117–128.
[32] Pan, Q.K., Ruiz, R., 2014. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega 44, 41–50.
[33] Potts, C., 1980. An adaptive branching rule for the permutation flow-shop problem. European Journal of Operational Research 5, 19–25.
[34] Reddy, D.R., et al., 1977. Speech understanding systems: A summary of results of the five-year research effort. department of computer science.
[35] Reza Hejazi*, S., Saghafian, S., 2005. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 43, 2895–2929.
[36] Ribas, I., Mateo, M., 2009. Improvement tools for neh based heuristics on permutation and blocking flow shop scheduling problems, in: IFIP International Conference on Advances in Production Management Systems, Springer. pp. 33–40.
[37] Ritt, M., 2016. A branch-and-bound algorithm with cyclic best-first search for the permutation flow shop scheduling problem, in: 2016 IEEE International Conference on Automation Science and Engineering (CASE), IEEE. pp. 872–877.
[38] Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049.
[39] Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. European journal of Operational research 47, 65–74.
[40] Taillard, E., 1993. Benchmarks for basic scheduling problems. european journal of operational research 64, 278–285.
[41] Vadlamudi, S.G., Gaurav, P., Aine, S., Chakrabarti, P.P., 2012. Anytime column search, in: Australasian Joint Conference on Artificial Intelligence, Springer. pp. 254–265.
[42] Vakharia, A.J., Wemmerlov, U., 1990. Designing a cellular manufacturing system: a materials flow approach based on operation sequences. IIE transactions 22, 84–97.
[43] Vallada, E., Ruiz, R., Framinan, J.M., 2015. New hard benchmark for flowshop scheduling problems minimising makespan. European Journal of Operational Research 240, 666–677.
[44] Vasiljevic, D., Danilovic, M., 2015. Handling ties in heuristics for the permutation flow shop scheduling problem. Journal of Manufacturing Systems 35, 1–9.
[45] Wang, L., Pan, Q.K., Tasgetiren, M.F., 2011. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. Computers & Industrial Engineering 61, 76–83.
[46] Zheng, D.Z., Wang, L., 2003. An effective hybrid heuristic for flow shop scheduling. The International Journal of Advanced Manufacturing Technology 21, 38–44.
[47] Zhou, R., Hansen, E.A., 2005. Beam-stack search: Integrating backtracking with beam search., in: ICAPS, pp. 90–98.

A. Notations

- J: all the jobs
- M: all the machines
- n: job number ($n = |J|$)
- m: machine number ($m = |M|$)
- F (resp. B): all the jobs scheduled in the prefix (resp. suffix)
- $Cmax_{f,i}$: first availability of machine i in the forward search
- $Cmax_{b,i}$: first availability of machine i in the backward search
- R_i: remaining processing time on machine i. $R_i = \sum_{j \in J \setminus \{F \cup B\}} p_{ij}$
- $I_{f,i}$: total idle time on machine i in the forward search
- $I_{b,i}$: total idle time on machine i in the backward search
- α: proportion of scheduled jobs. $\alpha = \frac{|F|+|B|}{|J|}$ on bi-directional branching or $\alpha = \frac{|F|}{|J|}$ on forward branching.
- g_{bound}: guidance function based on the bound (makespan or flowtime)
- g_{idle}: guidance function based only by the idle time
- g_{alpha}: guidance function based on both the bound and idle time
- g_{walpha}: guidance function based on both the bound and weighted idle time
- $g_{\text{wfrontalpha}}$: guidance function based on both the bound and the proportion of idle time in the partial solution
- g_{gap}: guidance function based on both the gap, bound, and weighted idle time

B. detailed numerical results

instance	Gmys_B&B	IGrms	VBIH	IGbob	IBS_gap	IBS_wfrontalpha
VFR100_20_1	**6.121**	6.176	6.173	6.178	6.252	6.170
VFR100_20_2	**6.224**	6.267	6.227	6.286	6.364	6.287
VFR100_20_3	**6.157**	6.210	6.264	6.216	6.319	6.232
VFR100_20_4	**6.173**	6.223	6.285	6.229	6.287	6.257
VFR100_20_5	**6.221**	6.260	6.401	6.270	6.425	6.322
VFR100_20_6	6.247	6.274	**6.074**	6.304	6.504	6.342
VFR100_20_7	6.358	6.411	**6.328**	6.416	6.575	6.412
VFR100_20_8	**6.023**	6.074	6.125	6.088	6.242	6.093
VFR100_20_9	6.286	6.324	**6.267**	6.329	6.476	6.347
VFR100_20_10	**6.048**	6.119	6.221	6.140	6.271	6.202
VFR100_40_1	-	**7.840**	7.846	7.844	8.263	7.980
VFR100_40_2	-	7.957	**7.913**	7.967	8.390	8.043
VFR100_40_3	-	**7.889**	7.997	7.905	8.308	7.979
VFR100_40_4	-	**7.895**	7.993	7.911	8.298	8.027
VFR100_40_5	-	**7.968**	7.980	8.002	8.444	8.087
VFR100_40_6	-	7.988	**7.957**	7.994	8.416	8.111
VFR100_40_7	-	7.956	**7.888**	7.980	8.409	8.163
VFR100_40_8	-	7.936	**7.917**	7.956	8.441	8.100
VFR100_40_9	-	**7.853**	7.976	7.882	8.354	7.978
VFR100_40_10	-	**7.894**	**7.894**	7.923	8.266	8.018
VFR100_60_1	-	**9.326**	9.353	9.350	10.051	9.677
VFR100_60_2	-	9.513	**9.403**	9.539	10.072	9.717
VFR100_60_3	-	**9.316**	9.431	9.332	9.908	9.601
VFR100_60_4	-	**9.366**	9.630	9.390	9.974	9.589
VFR100_60_5	-	9.391	**9.346**	9.404	9.993	9.639
VFR100_60_6	-	9.622	**9.523**	9.641	10.234	9.944
VFR100_60_7	-	**9.326**	9.488	9.358	9.970	9.635
VFR100_60_8	-	**9.507**	9.572	9.511	10.128	9.815
VFR100_60_9	-	**9.480**	9.567	9.494	10.141	9.823
VFR100_60_10	-	9.547	**9.349**	9.568	10.134	9.807
VFR200_20_1	**11.181**	11.271	11.272	11.283	11.260	11.416
VFR200_20_2	11.254	11.227	**11.188**	11.236	11.262	11.439
VFR200_20_3	11.233	11.297	**11.143**	11.294	11.523	11.373
VFR200_20_4	**11.090**	11.175	11.310	11.177	11.307	11.245
VFR200_20_5	**11.076**	11.152	11.365	11.147	11.292	11.285
VFR200_20_6	11.208	11.301	**11.128**	11.311	11.339	11.393
VFR200_20_7	11.266	11.347	**11.091**	11.356	11.438	11.439
VFR200_20_8	**11.041**	11.107	11.294	11.118	**11.041**	11.268
VFR200_20_9	**11.008**	11.069	11.240	11.074	11.308	11.254
VFR200_20_10	**11.193**	11.286	11.294	11.278	11.322	11.370
VFR200_40_1	-	**13.077**	13.124	13.084	13.794	13.107
VFR200_40_2	-	**13.027**	13.163	13.053	13.688	13.036
VFR200_40_3	-	13.197	**12.974**	13.216	13.832	13.194
VFR200_40_4	-	13.111	**13.061**	13.114	13.745	13.110
VFR200_40_5	-	**12.927**	13.220	12.957	13.601	12.945
VFR200_40_6	-	**13.023**	13.132	13.040	13.772	13.055
VFR200_40_7	-	13.188	**13.033**	13.204	14.099	13.204
VFR200_40_8	-	**13.089**	13.146	13.133	13.782	13.107
VFR200_40_9	-	**13.042**	13.049	13.077	13.782	13.051
VFR200_40_10	-	13.134	13.222	13.134	13.865	**13.091**
VFR200_60_1	-	**14.861**	14.906	14.884	16.102	15.020
VFR200_60_2	-	**14.890**	14.968	14.905	16.078	15.062
VFR200_60_3	-	15.103	**15.042**	15.141	16.462	15.446
VFR200_60_4	-	**14.918**	14.996	14.942	16.075	15.279
VFR200_60_5	-	15.020	**15.006**	15.030	16.258	15.348
VFR200_60_6	-	14.909	**14.894**	14.948	15.969	15.101
VFR200_60_7	-	14.956	**14.925**	14.991	16.170	15.179
VFR200_60_8	-	**14.852**	14.908	14.898	16.042	15.119
VFR200_60_9	-	**14.867**	14.909	14.900	15.972	15.167
VFR200_60_10	-	**14.881**	15.134	14.891	15.838	15.251

Figure 11: Makespan minimization full results: 100 and 200 jobs

instance	Gmys_B&B	IGrms	VBIH	IGbob	IBS_gap	IBS_wfrontalpha
VFR300_20_1	**15.996**	16.092	16.089	16.098	16.153	16.236
VFR300_20_2	16.409	16.465	**16.168**	16.463	16.470	16.648
VFR300_20_3	**16.010**	16.115	16.307	16.139	16.152	16.344
VFR300_20_4	**16.052**	16.125	16.095	16.149	16.060	16.309
VFR300_20_5	21.399	16.293	**16.244**	16.321	16.278	16.522
VFR300_20_6	**16.021**	16.062	16.369	16.063	**16.021**	16.320
VFR300_20_7	**16.188**	16.228	16.324	16.226	16.215	16.375
VFR300_20_8	**16.287**	16.363	16.798	16.371	16.545	16.516
VFR300_20_9	**16.203**	16.298	16.483	16.335	16.347	16.492
VFR300_20_10	16.780	16.794	**16.129**	16.794	16.780	17.075
VFR300_40_1	-	18.127	18.199	18.116	19.157	**18.056**
VFR300_40_2	-	18.341	18.227	18.330	19.199	**18.224**
VFR300_40_3	-	18.276	18.343	18.317	19.393	**18.249**
VFR300_40_4	-	18.181	18.340	18.263	19.467	**18.095**
VFR300_40_5	-	18.320	18.396	18.343	19.545	**18.198**
VFR300_40_6	-	18.250	18.290	18.318	19.444	**18.177**
VFR300_40_7	-	18.283	18.261	18.301	19.204	**18.202**
VFR300_40_8	-	18.238	18.286	18.237	19.149	**18.193**
VFR300_40_9	-	18.226	18.373	18.268	19.393	**18.093**
VFR300_40_10	-	18.253	18.348	18.230	19.037	**18.135**
VFR300_60_1	-	**20.397**	20.483	20.420	21.942	20.561
VFR300_60_2	-	**20.224**	20.293	20.250	21.722	20.444
VFR300_60_3	-	20.244	**20.200**	20.288	22.124	20.468
VFR300_60_4	-	20.235	20.280	**20.203**	21.711	20.477
VFR300_60_5	-	**20.156**	20.358	20.202	21.651	20.235
VFR300_60_6	-	**20.180**	20.319	20.254	21.691	20.427
VFR300_60_7	-	**20.285**	20.405	20.306	21.834	20.509
VFR300_60_8	-	**20.291**	20.385	20.293	21.959	20.668
VFR300_60_9	-	20.326	**20.249**	20.365	22.059	20.503
VFR300_60_10	-	**20.290**	20.328	20.345	22.075	20.524
VFR400_20_1	**20.952**	21.027	21.042	21.051	21.098	21.174
VFR400_20_2	21.346	21.411	**21.237**	21.432	21.527	21.586
VFR400_20_3	**21.379**	21.426	21.528	21.421	21.426	21.763
VFR400_20_4	**21.125**	21.231	21.188	21.226	21.171	21.492
VFR400_20_5	**16.245**	21.497	21.599	21.543	21.430	21.678
VFR400_20_6	**21.075**	21.165	21.264	21.177	21.289	21.278
VFR400_20_7	21.507	21.580	**21.293**	21.604	21.526	21.842
VFR400_20_8	**21.198**	21.264	21.526	21.261	21.216	21.455
VFR400_20_9	**21.236**	21.301	21.411	21.298	21.379	21.468
VFR400_20_10	21.456	21.524	**21.428**	21.524	21.456	21.678
VFR400_40_1	-	23.362	23.393	23.323	24.751	**23.139**
VFR400_40_2	-	23.257	23.269	23.274	24.529	**23.037**
VFR400_40_3	-	23.405	23.213	23.420	24.589	**23.202**
VFR400_40_4	-	23.220	23.298	23.211	24.925	**22.894**
VFR400_40_5	-	23.141	23.415	23.153	24.754	**22.947**
VFR400_40_6	-	23.292	23.290	23.318	24.603	**23.092**
VFR400_40_7	-	23.364	23.424	23.343	24.672	**23.184**
VFR400_40_8	-	23.266	23.606	23.228	24.633	**23.131**
VFR400_40_9	-	23.457	23.380	23.438	24.763	**23.323**
VFR400_40_10	-	23.504	23.467	23.535	24.597	**23.348**
VFR400_60_1	-	25.392	25.395	25.440	27.595	**25.292**
VFR400_60_2	-	25.498	25.638	25.525	27.667	**25.473**
VFR400_60_3	-	25.590	25.669	**25.541**	27.866	25.658
VFR400_60_4	-	25.608	**25.407**	25.658	27.927	25.813
VFR400_60_5	-	25.615	**25.415**	25.554	27.565	25.697
VFR400_60_6	-	25.358	25.603	**25.350**	27.441	25.398
VFR400_60_7	-	**25.372**	25.673	25.394	27.643	25.404
VFR400_60_8	-	25.541	25.658	25.593	27.632	**25.469**
VFR400_60_9	-	25.622	**25.549**	25.672	27.443	25.622
VFR400_60_10	-	25.618	25.707	25.631	27.797	**25.556**

Figure 12: Makespan minimization full results: 300 and 400 jobs

instance	Gmys_B&B	IGrms	VBIH	IGbob	IBS_gap	IBS_wfrontalpha
VFR500_20_1	**26.253**	26.355	26.374	26.351	26.356	26.563
VFR500_20_2	26.555	26.631	**26.080**	26.638	26.708	26.869
VFR500_20_3	**26.268**	26.357	26.759	26.356	26.344	26.597
VFR500_20_4	**25.994**	26.058	26.411	26.076	26.009	26.316
VFR500_20_5	26.703	26.729	**26.409**	26.729	26.727	27.108
VFR500_20_6	26.325	26.395	**26.305**	26.402	26.325	26.558
VFR500_20_7	**26.313**	26.401	26.430	26.401	26.438	26.680
VFR500_20_8	26.217	26.302	**26.034**	26.302	26.327	26.496
VFR500_20_9	**26.345**	26.410	26.641	26.419	26.405	26.650
VFR500_20_10	26.345	26.043	26.359	26.055	**26.024**	26.323
VFR500_40_1	-	28.362	28.402	28.353	30.321	**28.129**
VFR500_40_2	-	28.503	28.615	28.522	30.012	**28.308**
VFR500_40_3	-	28.374	28.579	28.442	29.620	**28.235**
VFR500_40_4	-	28.477	28.432	28.530	29.824	**28.329**
VFR500_40_5	-	28.543	28.553	28.518	30.019	**28.283**
VFR500_40_6	-	28.248	28.488	28.336	29.535	**28.134**
VFR500_40_7	-	28.486	28.640	28.551	29.900	**28.323**
VFR500_40_8	-	28.435	28.644	28.444	30.130	**28.307**
VFR500_40_9	-	28.640	28.613	28.610	30.351	**28.406**
VFR500_40_10	-	28.585	28.526	28.582	29.784	**28.324**
VFR500_60_1	-	30.609	30.682	30.649	33.168	**30.429**
VFR500_60_2	-	30.597	30.793	30.608	32.593	**30.480**
VFR500_60_3	-	30.823	30.763	30.799	33.506	**30.553**
VFR500_60_4	-	30.796	30.788	30.740	32.712	**30.672**
VFR500_60_5	-	30.700	30.826	30.727	33.339	**30.540**
VFR500_60_6	-	30.829	30.837	30.795	33.635	**30.597**
VFR500_60_7	-	30.733	30.805	30.680	32.988	**30.528**
VFR500_60_8	-	30.729	30.866	30.738	33.551	**30.465**
VFR500_60_9	-	30.785	30.664	30.777	33.224	**30.515**
VFR500_60_10	-	30.828	30.852	30.809	33.044	**30.787**
VFR600_20_1	**31.303**	31.359	31.372	31.354	**31.303**	31.525
VFR600_20_2	**31.281**	31.372	31.487	31.369	31.317	31.688
VFR600_20_3	**31.374**	31.412	31.407	31.412	**31.374**	31.676
VFR600_20_4	**31.417**	31.480	31.696	31.491	31.440	31.733
VFR600_20_5	**31.323**	31.387	31.527	31.389	31.476	31.659
VFR600_20_6	31.613	31.668	**31.523**	31.669	31.615	31.983
VFR600_20_7	**31.461**	31.483	31.532	31.527	**31.461**	31.893
VFR600_20_8	31.414	31.465	**31.107**	31.483	31.428	31.709
VFR600_20_9	31.473	31.514	**31.397**	31.515	31.554	31.971
VFR600_20_10	**31.021**	31.107	31.429	31.107	**31.021**	31.310
VFR600_40_1	-	33.618	33.683	33.600	35.743	**33.339**
VFR600_40_2	-	33.356	33.584	33.311	34.981	**33.200**
VFR600_40_3	-	33.612	**33.401**	33.576	35.364	33.415
VFR600_40_4	-	33.477	33.626	33.502	35.517	**33.235**
VFR600_40_5	-	33.307	33.545	33.280	34.823	**33.188**
VFR600_40_6	-	33.552	**33.298**	33.563	35.094	33.422
VFR600_40_7	-	33.492	33.567	33.495	35.679	**33.409**
VFR600_40_8	-	33.282	33.473	33.279	34.878	**33.068**
VFR600_40_9	-	33.422	33.405	33.441	35.346	**33.270**
VFR600_40_10	-	33.396	33.713	33.364	35.186	**33.308**
VFR600_60_1	-	35.863	35.976	35.862	38.094	**35.504**
VFR600_60_2	-	35.791	36.000	35.814	38.503	**35.450**
VFR600_60_3	-	35.896	36.004	35.940	38.501	**35.670**
VFR600_60_4	-	35.883	35.943	35.833	38.457	**35.610**
VFR600_60_5	-	35.929	35.965	35.880	38.553	**35.466**
VFR600_60_6	-	35.828	35.894	35.844	38.848	**35.617**
VFR600_60_7	-	35.882	35.987	35.952	38.517	**35.741**
VFR600_60_8	-	35.784	35.943	35.887	38.558	**35.368**
VFR600_60_9	-	35.935	35.923	35.882	39.466	**35.693**
VFR600_60_10	-	35.804	35.917	35.916	38.297	**35.597**

Figure 13: Makespan minimization full results: 500 and 600 jobs

instance	Gmys_B&B	IGrms	VBIH	IGbob	IBS_gap	IBS_wfrontalpha
VFR700_20_1	**36.285**	36.354	36.388	36.360	36.294	36.760
VFR700_20_2	**36.220**	36.303	36.380	36.316	36.244	36.726
VFR700_20_3	**36.419**	36.487	36.556	36.487	36.547	36.843
VFR700_20_4	**36.361**	36.379	36.645	36.384	**36.361**	36.788
VFR700_20_5	**36.496**	36.547	36.597	36.547	**36.496**	36.961
VFR700_20_6	36.556	36.610	**36.492**	36.615	36.556	36.926
VFR700_20_7	36.540	36.609	**36.315**	36.612	36.540	36.903
VFR700_20_8	36.418	36.481	**36.386**	36.465	36.418	36.775
VFR700_20_9	**36.212**	36.290	36.316	36.277	36.222	36.610
VFR700_20_10	**36.362**	36.376	36.519	36.398	**36.362**	36.809
VFR700_40_1	-	38.720	38.767	38.674	40.414	**38.550**
VFR700_40_2	-	38.499	38.597	38.524	40.378	**38.291**
VFR700_40_3	-	38.393	38.490	38.392	40.543	**38.141**
VFR700_40_4	-	38.593	38.440	38.585	40.522	**38.430**
VFR700_40_5	-	38.430	38.355	38.452	39.973	**38.267**
VFR700_40_6	-	38.336	38.817	38.350	39.825	**38.291**
VFR700_40_7	-	38.287	38.569	38.298	40.051	**38.058**
VFR700_40_8	-	38.766	38.712	38.735	40.377	**38.693**
VFR700_40_9	-	38.452	38.560	38.503	39.661	**38.413**
VFR700_40_10	-	38.647	**38.460**	38.598	40.042	38.566
VFR700_60_1	-	41.125	41.192	41.097	44.623	**40.615**
VFR700_60_2	-	41.008	41.120	40.994	43.069	**40.664**
VFR700_60_3	-	40.961	41.167	40.957	44.120	**40.581**
VFR700_60_4	-	41.070	41.159	40.997	44.001	**40.491**
VFR700_60_5	-	41.022	40.734	41.002	44.522	**40.650**
VFR700_60_6	-	40.994	41.305	40.983	44.849	**40.472**
VFR700_60_7	-	40.572	41.111	40.584	43.977	**40.171**
VFR700_60_8	-	41.121	41.186	41.140	44.205	**40.797**
VFR700_60_9	-	40.930	41.002	40.945	44.642	**40.421**
VFR700_60_10	-	41.093	41.173	41.083	44.467	**40.682**
VFR800_20_1	**41.413**	41.477	41.479	41.514	41.433	41.769
VFR800_20_2	**41.282**	41.337	41.426	41.337	41.286	41.613
VFR800_20_3	**41.319**	41.362	41.705	41.370	**41.319**	41.581
VFR800_20_4	**41.375**	41.426	41.961	41.426	41.443	41.919
VFR800_20_5	41.626	41.702	**41.395**	41.705	41.626	41.948
VFR800_20_6	41.919	41.959	**41.435**	41.957	41.919	42.375
VFR800_20_7	**41.342**	41.379	41.783	41.394	41.352	41.715
VFR800_20_8	**41.390**	41.429	41.568	41.430	41.539	41.951
VFR800_20_9	41.697	41.753	**41.345**	41.753	41.697	42.035
VFR800_20_10	41.489	41.561	**41.399**	41.565	41.489	41.942
VFR800_40_1	-	43.456	43.466	43.435	45.354	**43.221**
VFR800_40_2	-	43.483	43.743	43.516	45.309	**43.326**
VFR800_40_3	-	43.512	43.794	43.461	45.254	**43.255**
VFR800_40_4	-	43.557	43.638	43.632	44.981	**43.499**
VFR800_40_5	-	43.635	**43.484**	43.639	45.670	43.581
VFR800_40_6	-	43.549	43.666	43.549	45.459	**43.256**
VFR800_40_7	-	43.458	43.643	43.438	45.455	**43.311**
VFR800_40_8	-	43.548	43.630	43.555	44.822	**43.387**
VFR800_40_9	-	43.497	43.575	43.517	45.269	**43.389**
VFR800_40_10	-	43.592	43.596	43.567	45.091	**43.392**
VFR800_60_1	-	46.130	46.279	46.103	49.987	**45.683**
VFR800_60_2	-	46.164	46.261	46.167	49.777	**45.714**
VFR800_60_3	-	46.108	46.164	46.099	49.532	**45.627**
VFR800_60_4	-	46.035	46.288	46.157	49.545	**45.558**
VFR800_60_5	-	46.101	46.061	46.085	49.600	**45.614**
VFR800_60_6	-	46.110	46.257	46.124	50.056	**45.477**
VFR800_60_7	-	45.986	46.279	46.003	49.273	**45.524**
VFR800_60_8	-	46.136	46.211	46.206	49.679	**45.510**
VFR800_60_9	-	46.226	46.232	46.229	50.036	**45.808**
VFR800_60_10	-	46.004	46.258	45.995	50.099	**45.383**

Figure 14: Makespan minimization full results: 700 and 800 jobs

instance	ALGirtct	shake-LS	IBS_alpha	IBS_walpha
TA1 / TA20_5_0	**14.033**	**14.033**	**14.033**	**14.033**
TA2 / TA20_5_1	**15.151**	**15.151**	**15.151**	**15.151**
TA3 / TA20_5_2	**13.301**	**13.301**	**13.301**	13.313
TA4 / TA20_5_3	**15.447**	**15.447**	**15.447**	**15.447**
TA5 / TA20_5_4	**13.529**	**13.529**	**13.529**	**13.529**
TA6 / TA20_5_5	**13.123**	**13.123**	**13.123**	**13.123**
TA7 / TA20_5_6	**13.548**	**13.548**	**13.548**	**13.548**
TA8 / TA20_5_7	**13.948**	**13.948**	**13.948**	**13.948**
TA9 / TA20_5_8	**14.295**	**14.295**	**14.295**	**14.295**
TA10 / TA20_5_9	**12.943**	**12.943**	**12.943**	**12.943**
TA11 / TA20_10_0	**20.911**	**20.911**	**20.911**	**20.911**
TA12 / TA20_10_1	**22.440**	**22.440**	**22.440**	22.652
TA13 / TA20_10_2	**19.833**	**19.833**	**19.833**	19.877
TA14 / TA20_10_3	**18.710**	**18.710**	**18.710**	18.779
TA15 / TA20_10_4	**18.641**	**18.641**	**18.641**	**18.641**
TA16 / TA20_10_5	**19.245**	**19.245**	**19.245**	19.414
TA17 / TA20_10_6	**18.363**	**18.363**	**18.363**	18.462
TA18 / TA20_10_7	**20.241**	**20.241**	**20.241**	20.268
TA19 / TA20_10_8	**20.330**	**20.330**	**20.330**	20.481
TA20 / TA20_10_9	**21.320**	**21.320**	**21.320**	21.420
TA21 / TA20_20_0	**33.623**	**33.623**	**33.623**	33.638
TA22 / TA20_20_1	**31.587**	**31.587**	**31.587**	31.785
TA23 / TA20_20_2	**33.920**	**33.920**	**33.920**	34.318
TA24 / TA20_20_3	**31.661**	**31.661**	**31.661**	**31.661**
TA25 / TA20_20_4	**34.557**	**34.557**	**34.557**	34.726
TA26 / TA20_20_5	**32.564**	**32.564**	**32.564**	32.988
TA27 / TA20_20_6	**32.922**	**32.922**	**32.922**	33.199
TA28 / TA20_20_7	**32.412**	**32.412**	**32.412**	32.688
TA29 / TA20_20_8	**33.600**	**33.600**	**33.600**	34.235
TA30 / TA20_20_9	**32.262**	**32.262**	**32.262**	32.698
TA31 / TA50_5_0	**64.802**	**64.802**	65.207	64.904
TA32 / TA50_5_1	**68.051**	**68.051**	68.149	68.096
TA33 / TA50_5_2	**63.162**	**63.162**	63.247	**63.162**
TA34 / TA50_5_3	**68.226**	**68.226**	68.242	**68.226**
TA35 / TA50_5_4	**69.351**	**69.351**	69.895	69.460
TA36 / TA50_5_5	**66.841**	**66.841**	66.910	**66.841**
TA37 / TA50_5_6	**66.253**	**66.253**	66.427	66.277
TA38 / TA50_5_7	**64.332**	**64.332**	64.471	64.426
TA39 / TA50_5_8	**62.981**	**62.981**	63.878	63.212
TA40 / TA50_5_9	**68.770**	**68.770**	68.895	68.834

Figure 15: Flowtime minimization full results: TAI1 to TAI40

instance	ALGirtct	shake-LS	IBS_alpha	IBS_walpha
TA41 / TA50_10_0	**87.114**	**87.114**	87.183	87.413
TA42 / TA50_10_1	**82.820**	**82.820**	82.967	83.548
TA43 / TA50_10_2	**79.931**	**79.931**	80.148	80.411
TA44 / TA50_10_3	**86.446**	**86.446**	86.609	86.661
TA45 / TA50_10_4	**86.377**	**86.377**	86.567	86.628
TA46 / TA50_10_5	**86.587**	**86.587**	86.729	87.025
TA47 / TA50_10_6	**88.750**	**88.750**	89.739	89.867
TA48 / TA50_10_7	**86.727**	**86.727**	87.078	87.749
TA49 / TA50_10_8	**85.441**	**85.441**	85.952	86.532
TA50 / TA50_10_9	**87.998**	**87.998**	88.546	88.967
TA51 / TA50_20_0	**125.831**	**125.831**	125.850	126.406
TA52 / TA50_20_1	**119.247**	**119.247**	119.463	120.630
TA53 / TA50_20_2	**116.459**	**116.459**	116.536	118.533
TA54 / TA50_20_3	**120.261**	**120.261**	121.035	121.614
TA55 / TA50_20_4	**118.184**	**118.184**	118.379	119.974
TA56 / TA50_20_5	**120.586**	**120.586**	120.897	121.671
TA57 / TA50_20_6	**122.880**	**122.880**	123.120	124.423
TA58 / TA50_20_7	**122.489**	**122.489**	122.583	124.033
TA59 / TA50_20_8	**121.872**	**121.872**	**121.872**	123.347
TA60 / TA50_20_9	**123.954**	**123.954**	124.458	125.425
TA61 / TA100_5_0	253.167	253.167	252.863	**252.780**
TA62 / TA100_5_1	241.989	241.925	**241.738**	241.858
TA63 / TA100_5_2	237.832	237.832	**237.331**	237.412
TA64 / TA100_5_3	227.738	227.522	228.013	**227.335**
TA65 / TA100_5_4	240.301	240.301	**240.114**	240.144
TA66 / TA100_5_5	232.247	232.342	232.177	**232.078**
TA67 / TA100_5_6	240.366	240.366	240.790	**239.994**
TA68 / TA100_5_7	230.866	230.945	**230.328**	230.405
TA69 / TA100_5_8	247.771	247.526	**247.478**	247.611
TA70 / TA100_5_9	**242.933**	**242.933**	243.710	243.156
TA71 / TA100_10_0	**298.385**	**298.385**	298.578	299.198
TA72 / TA100_10_1	273.674	273.674	273.852	**273.282**
TA73 / TA100_10_2	288.114	288.114	288.302	**287.614**
TA74 / TA100_10_3	301.044	301.044	**300.738**	300.818
TA75 / TA100_10_4	284.148	284.233	**283.961**	284.023
TA76 / TA100_10_5	269.686	269.686	269.672	**269.664**
TA77 / TA100_10_6	**279.463**	**279.463**	281.049	280.196
TA78 / TA100_10_7	290.703	290.908	**290.252**	290.856
TA79 / TA100_10_8	301.970	301.970	**301.967**	302.783
TA80 / TA100_10_9	**291.283**	**291.283**	291.566	291.293

Figure 16: Flowtime minimization full results: TAI41 to TAI80

instance	ALGirtct	shake-LS	IBS_alpha	IBS_walpha
TA81 / TA100_20_0	**365.463**	**365.463**	366.625	368.870
TA82 / TA100_20_1	372.001	372.449	**371.544**	374.294
TA83 / TA100_20_2	370.027	370.027	**369.571**	375.822
TA84 / TA100_20_3	372.393	372.393	**371.683**	375.772
TA85 / TA100_20_4	368.915	368.915	**368.393**	371.224
TA86 / TA100_20_5	**370.908**	**370.908**	370.953	376.096
TA87 / TA100_20_6	373.408	373.408	**372.606**	376.264
TA88 / TA100_20_7	384.525	384.525	**384.292**	387.387
TA89 / TA100_20_8	374.423	374.423	**374.413**	378.309
TA90 / TA100_20_9	379.296	379.296	**378.948**	381.752
TA91 / TA200_10_0	1.042.452	1.041.023	1.041.139	**1.035.643**
TA92 / TA200_10_1	1.028.775	1.028.828	1.026.655	**1.025.575**
TA93 / TA200_10_2	1.043.631	1.042.357	1.043.126	**1.040.027**
TA94 / TA200_10_3	1.023.188	1.025.564	1.023.864	**1.019.733**
TA95 / TA200_10_4	1.028.506	1.028.963	1.030.206	**1.023.655**
TA96 / TA200_10_5	998.686	998.340	996.911	**994.767**
TA97 / TA200_10_6	1.042.570	1.042.570	1.041.190	**1.039.256**
TA98 / TA200_10_7	1.035.945	1.035.915	1.035.240	**1.034.400**
TA99 / TA200_10_8	1.015.560	1.015.280	1.015.094	**1.013.493**
TA100 / TA200_10_9	1.021.633	1.021.865	1.019.093	**1.017.843**
TA101 / TA200_20_0	1.221.768	1.219.341	**1.213.435**	1.225.238
TA102 / TA200_20_1	**1.231.880**	1.233.161	1.232.137	1.233.707
TA103 / TA200_20_2	1.254.822	1.259.605	**1.253.345**	1.256.823
TA104 / TA200_20_3	1.226.654	1.228.027	**1.223.157**	1.224.065
TA105 / TA200_20_4	1.215.411	1.215.854	**1.211.625**	1.215.865
TA106 / TA200_20_5	1.219.698	1.218.757	**1.213.883**	1.218.650
TA107 / TA200_20_6	1.237.014	**1.234.330**	1.235.129	1.237.112
TA108 / TA200_20_7	1.233.257	1.240.105	**1.232.346**	1.235.926
TA109 / TA200_20_8	1.222.431	1.220.058	**1.218.417**	1.221.633
TA110 / TA200_20_9	**1.234.864**	1.235.113	1.235.641	1.241.081
TA111 / TA500_20_0	6.562.522	6.558.109	6.552.189	**6.547.180**
TA112 / TA500_20_1	6.678.713	6.679.339	6.675.497	**6.662.028**
TA113 / TA500_20_2	6.632.299	6.624.644	6.623.513	**6.603.783**
TA114 / TA500_20_3	6.633.622	6.646.006	6.636.420	**6.616.575**
TA115 / TA500_20_4	6.609.322	6.587.110	6.587.941	**6.580.659**
TA116 / TA500_20_5	6.605.982	6.602.685	6.595.286	**6.591.712**
TA117 / TA500_20_6	6.576.412	6.576.047	6.568.221	**6.563.446**
TA118 / TA500_20_7	6.628.915	6.629.065	6.617.381	**6.616.946**
TA119 / TA500_20_8	6.569.013	6.587.638	6.576.528	**6.568.454**
TA120 / TA500_20_9	6.614.629	6.623.849	6.624.923	**6.598.425**

Figure 17: Flowtime minimization full results: TAI81 to TAI120