Adaptive Bias Generalized Rollout Policy Adaptation on the Flexible Job-Shop Scheduling Problem

Lotfi Kobrosly[1,2], Marc-Emmanuel Coupvent des Graviers[1], Christophe Guettier[1], and Tristan Cazenave[2]

[1] Safran Electronics and Defense, France
[2] LAMSADE, Université Paris Dauphine-PSL, Place du Maréchal de Lattre de Tassigny, Paris, France

Abstract. The Flexible Job-Shop Scheduling Problem (FJSSP) is an NP-hard combinatorial optimization problem, with several application domains, especially for manufacturing purposes. The objective is to efficiently schedule multiple operations on dissimilar machines. These operations are gathered into jobs, and operations pertaining to the same job need to be scheduled sequentially. Different methods have been previously tested to solve this problem, such as Constraint Solving, Tabu Search, Genetic Algorithms, or Monte Carlo Tree Search (MCTS). We propose a novel algorithm derived from the Generalized Nested Rollout Policy Adaptation, developed to solve the FJSSP. We report encouraging experimental results, as our algorithm performs better than other MCTS-based approaches, even if makespans obtained on large instances are still far from known upper bounds.

Keywords: Monte Carlo Tree Search · Generalized Nested Rollout Policy Adaptation · Policy Gradient Descent · Markov Decision Process · Flexible Job-Shop Scheduling Problem

1 Introduction

The Flexible Job-Shop Scheduling Problem (FJSSP), as an extension of the more widely known Job-Shop Scheduling Problem (JSSP), has been studied for years in the research community. This is due to its proximity to manufacturing, aeronautics, and medicine applications [46,20], but also because of the computational challenge it poses, since it qualifies as an NP-hard problem [43,55,53,37]. The usual branch-and-bound methods that work for the JSSP do not generalize well to the FJSSP [20], suggesting the need for alternative methods to solve this problem.

The goal of the FJSSP is to assign ordered operations of different jobs sequentially on the available machines, while minimizing a certain objective function[23]. To address the problem, Monte Carlo Tree Search (MCTS), which can be considered as a form of Reinforcement Learning [51] that quantifies the tradeoff between exploration and exploitation through probing [7], qualifies as

a promising solving method. Indeed, we can consider the FJSSP as a sequential decision problem. MCTS methods proved successful in solving this type of problems, such as one-player and two-player games [10,49] or in Operational Research (OR) problems like the Vehicle Routing Problem [13].

In this paper, we present a new approach called the *Adaptive Bias Generalized Nested Rollout Policy Adaptation (ABGNRPA)* on the FJSSP and compare it to the Generalized Nested Rollout Policy Adaptation (GNRPA) from which it stems, as well as to other variations of the MCTS. Section 2 describes the FJSSP problem and the GNRPA while presenting some of the previous work on these subjects. Section 3 presents the ABGNRPA and finally section 4 showcases our experimental procedure and results.

2 Background and motivation

2.1 The Job-Shop Scheduling Problem

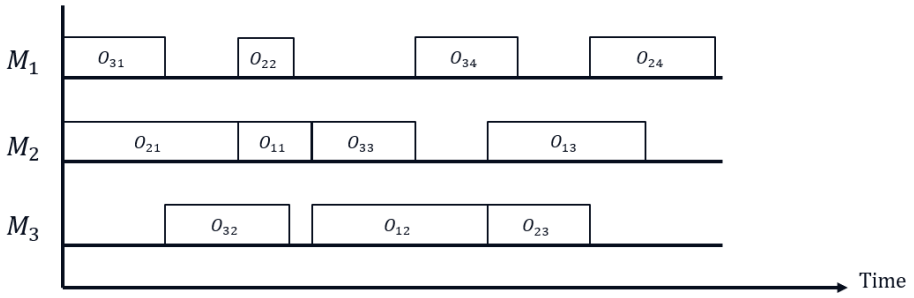

Fig. 1: Gantt's diagram of an example of a JSSP instance. The operations denoted by O_{ij} are assigned to machines M_l while respecting the constraints of order of operations in a job, of compatibility with machines and no interruption of the processing of operations. Blank spaces in a machine's line of processing represent inactivity.

The JSSP is a combinatorial optimization and scheduling problem. It is a theoretical representation of several industrial use cases where we schedule the processing of several operations in an orderly fashion to optimize a cost function. The formalized setting of the JSSP can be defined as follows, with a visual representation available in figure 1:

- A set of n jobs $\mathcal{J} = \{\mathcal{J}_i \mid i \in \{1..n\}\}$.
- A set of m machines, $\mathcal{M} = \{m_l, l \in \{1..m\}\}$.
- For every job $\mathcal{J}_i,\ i \in \{1..n\}$, there are n_i operations $\mathcal{J}_i = \{o_{ij},\ j \in \{1..n_i\}\}$.
- The operations in each job are ordered, which means that for operations $o_{ij}, o_{ik} \in \mathcal{J}_i,\ j,k \in \{1..n_i\}$ and $j < k$, o_{ij} must be processed before o_{ik}.

- Every operation o_{ij} is defined in the problem's instance to be compatible with only one machine $m_l,\ l \in \{1..m\}$ with a fixed processing time d_{ij}.
- Once the processing of an operation o_{ij} starts on a machine m_l, it is done until its completion, *i.e* it cannot be interrupted.
- The objective is to optimize a cost function.

Several approaches have been adopted to address JSSP. Some of them model it as a Mixed Integer Linear Program [39], while others considered solving it with Constraint Programming (CP) [59,18], Tabu Search [41] or other local search techniques [3]. More recently, Deep Reinforcement Learning (DRL) has been used as one of the most prominent approaches considering the JSSP as a sequential decision problem [34,60]. Graph Neural Networks have gained a good amount of consideration, whether used individually or combined with DRL [25] or a randomized ϵ-greedy that can select the second best action [1].

2.2 The Flexible Job-Shop Scheduling Problem

The FJSSP is an extension of the JSSP, where in addition to the aforementioned setting, the assignment of operations on machines becomes an extra decision. Indeed, each operation can be processed by more than one machine instead of being statically allocated on a unique one as described in section 2.1. Every couple of operation and machine (o_{ij}, m_l) is associated with a processing time d_{ijl} if they are compatible. These additions enlarge the search space as the choice of the assignment of an operation on a machine changes the duration, remove homogeneity within the problem structure, for instance where symmetry breaking is harder to apply, and therefore make the problem more difficult to approach. Solving it requires assigning operations to machines, on top of ordering them and setting their start times. The objective can vary from the most studied criterion of the minimization of the makespan *i.e*, the total execution time [20] to other ones *e.g* the minimization of the workload at all instants t [16] or the completion times of jobs [21]. In our case, we focus on makespan minimization, which is more documented, providing us with more benchmarks to compare with our results.

The FJSSP can be represented with different approaches in addition to some of those applicable to the JSSP [20,30,40]. Several solving methods have been developed ranging from using Representation Learning via Graph Neural Networks [42,50] to Deep Reinforcement Learning [57], Memetic Algorithm [36] and Genetic Algorithm [21], considering the problem as a Markov Decision Process (MDP) [5,42]. For the Dynamic FJSSP, another variant of the FJSSP, a time-based approach is adopted as operations and jobs can be added during the processing of already existing operations [60,58].

2.3 Generalized Nested Rollout Policy Adaptation

Stochastic methods are useful for searching for good solutions, typically on an NP-hard problem such as the FJSSP. We can consider the FJSSP as a sequential decision making problem modeled by a search tree (see section 4.1 for the

modeling details). Thus, a MCTS approach is a good candidate, in particular, the Generalized Nested Rollout Policy Adaptation (GNRPA) [9]. In the search tree, a node represents a state of the advancement of a solution, and an arc from a parent node to a child represents an action.

In the general setting of a GNRPA, a rollout, or a playout, is one path through the search tree, from its root to a leaf. The choice of actions to generate this path is operated using a policy [33] that assigns a weight to every action independently from the state. The weight defines the probability of selecting the corresponding action from available ones in the current state, using a *softmax with temperature* function. The state-independent feature means that even if the set of possible actions differs from one state to another, an action has the same weight in all action sets in which it appears. Changes that occur to this weight are propagated to other action sets. The informal description of a *level 1* GNRPA is as follows [9]:

1. The policy is initialized with the same value for all actions.
2. A rollout is conducted through the search tree using the policy as a weighted random choice on the actions' set to select its next move. For every action, we compute a *bias* β_{action} statically via a *heuristic* [47]. For the FJSSP, we would refer to them as β_{ijl} for job i, operation j and machine l. This bias helps to initialize the probabilities of the actions.
3. At every state, we compute the probabilities of the possible actions, according to the formula in equation 1, and the weighted randomized choice is performed on a softmax computation with τ a *temperature*.
4. At the end of this playout, we compare to the last best sequence and take the best out of the two based on their respective scores. Through this sequence, the weights of the encountered possible actions are updated, as shown in equation 2, where α is a learning rate. $\delta_{aa'} = 1$ for the chosen action in the trajectory $a = a'$ (assigning operation j of job i on machine l), thus increases its weight, and $\delta_{aa'} = 0$ otherwise decreasing those weights. This changes the probability distribution exploiting the results of the playouts.
5. The new policy is then used for the following playout, as well as the biases.
6. This process is repeated for $n_{policies}$, through which the policy is updated at each iteration and used in the next one, whether a better solution is found, in which case, its actions' weights are increased, or the actions' weights of the last best solution are increased.

$$p_{a,s} = \frac{e^{\frac{w_a}{\tau}+\beta_a}}{\sum_{a'} e^{\frac{w_{a'}}{\tau}+\beta_{a'}}} \tag{1}$$

$$w_a \leftarrow w_a - \alpha\frac{p_{a,s} - \delta_{aa'}}{\tau} \tag{2}$$

If we consider a GNRPA with bias-free policy, we get a simple Nested Rollout Policy Adaptation (NRPA) algorithm [45,9] which follows the same procedure. For a level 2 GNRPA, we consider *two levels of nesting*. At the first level, a policy

is initialized with the biases. Then, at the second level, a GNRPA is conducted as described above for $n_{nested\_policies}$ iterations using a copy of the initial policy. At the end of these iterations, the best sequence is used to update the weights of the first level policy, regardless of the weights of the nested policy resulting at the second level. We then use this recently updated policy of level 1 for a second run of a lower level GNRPA. From this run, we also take the best sequence and update the higher level policy, and so on. Using more levels of nesting increases the computation time exponentially, with GNRPA depending on the depth of the search tree and the number of nested policies.

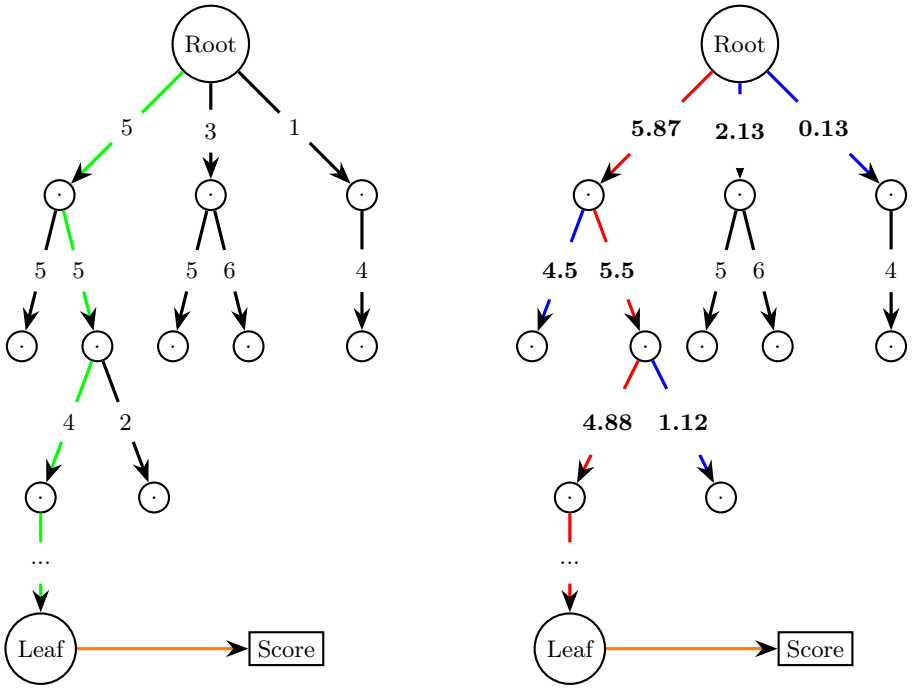

Fig. 2: Representation of a Nested Rollout Policy Adaptation, also compatible with GNRPA. The figure on the left represents a rollout, and arcs in green represent the chosen actions at each step. In this figure, the sequence obtained is the best one found so far. We use it to adapt the policy, as described in section 2.3. Otherwise, we use the last best sequence for the policy update. Then we run the next rollout shown on the right. The values in bold are the ones that were updated, the incremented ones have their arcs in red, belonging to the best sequence and the decremented ones are in blue.

2.4 Motivation

Improving the total makespan of an industrial assembly line has a clear impact on the productivity and cost-efficiency, which highlights the importance of FJSSP. Exact methods are effective in providing optimal or near-optimal results [39,59,18]. However, they can struggle in some extensions of the FJSSP where operations can be added dynamically [44,15], transportation times between machines are considered [29] or when uncertainties arise regarding processing times [27], machine capacity [2], machine failures or cost fluctuation [38].

MCTS-based methods have proven their efficacy in dealing with several problems and provide state-of-the-art results on games like Go [49] and Chess [17] or biology applications like inverse RNA folding [56] as well as energy systems, graph navigation, and, most importantly for us, combinatorial optimization problems [31]. These methods require little computational and memory resources, rendering them compatible with embedded applications.

To the best of our knowledge, only a few attempts use MCTS to solve the FJSSP or its variants [46,54,32,35] and they show promise. Yet, they only rely on classic MCTS and do not leverage the more effective potential of its more recent variants such as GNRPA shown in other works [45,8,47] where the information is mostly held by the choice of action independently from the state. We do highlight that this characteristic does not generalize to all problems such as navigation and some games where states contain most of the information with a limited set of possible actions in every state. In addition, previous research only focused on generated or private instances related to industrial needs or on a small number of public benchmarks. Thus, we investigate the performance of these methods in this particular setting on more public benchmark datasets.

We have initially conducted a relevance study, available in the appendix, to assess the usefulness of using such algorithms on this particular problem. Furthermore, we propose a novel variant that relies on the heuristic used to compute the biases. Our intuition is that adapting the biases throughout the playouts will provide a more adequate prior for the considered instance. Although it can still converge to a local minimum, this minimum might prove to be better than what previous variations of the MCTS can provide.

2.5 Disclaimer

Our objective is not necessarily to outperform state-of-the-art methods, given that CP-models are more effective than search-based ones here. Indeed, the results available in the literature and to which we are comparing ours are given mostly by CP-based models [20]. However, we aim to improve the performance of existing MCTS algorithms and showcase it on an academic problem with several benchmarks.

3 The Adaptive Bias Generalized Nested Rollout Policy Adaptation (ABGNRPA)

Our main contribution is an enhancement of GNRPA, called ABGNRPA. Compared to GNRPA, after initializing the biases in the same way, we **update them at each step of the playout**. This forces the bias to be dynamic and to rely on the results of the exploration for a better estimation of the potential of the trajectory. This differs from the update of the weights of the policy which happens at the end of the trajectory using a reward, *i.e* the makespan, whereas biases are updated at every step of every playout.

Application to FJSSP

The GNRPA and ABGNRPA biases are initialized using the same heuristic called *Earliest Ending Time*. For every action, we compute the minimal starting time possible of processing the operation, given the current state, and add the duration corresponding to the couple (operation, machine). The operations finishing earlier are given a higher value as in equation 3. We also normalize the biases taking into account all possible actions, see equation 4. The choice of this heuristic is based on experimental results as it gives a better overall performance compared to other ones, as per the appendix, table 5. The superiority of this heuristic appears to come from the facts that it is optimistic and captures some information from the current state.

The update of the biases, which is specific to ABGNRPA, is based on a FJSSP makespan lower bound ($\mathcal{LB}$). It is shown in equation 5 where i and j refer respectively to the job and operation, l to the machine, $\mathcal{LB}_{ijl}$ is the makespan lower bound value in the current state, γ is a learning rate, and $\delta_{ijl} = 1$ if assigning operation j of job i on machine l is chosen or $\delta_{ijl} = -\frac{1}{n_{actions}}$ otherwise.

The $\mathcal{LB}$ is the maximum remaining time on all jobs $i \in \{1..n\}$, added to the current makespan at time t, $\mathcal{M}_t$. For every job, we compute the sum of the minimal durations of every operation that remains, which are represented by the set $\mathcal{U} = \{o_{i,j}, i \in \{1..n\}$ where $o_{i,j}$ has not been assigned to a machine yet$\}$ and take the maximum value obtained, as detailed in equation 6. These updated biases are then used for the following playouts, as shown in algorithm 1. We believe we can enlarge the application of this algorithm to other problems given a relevant lower bound function.

$$EET_{ijl} \leftarrow -(earliest\_start\_time(action_{ijl}, state) + d_{ijl}) \tag{3}$$

$$\beta_{ijl} \leftarrow \frac{EET_{ijl}}{\sum_{action} EET_{action}} \tag{4}$$

$$\beta_{ijl} \leftarrow \beta_{ijl} + \gamma * \delta_{ijl} * \mathcal{LB}_{ijl} \tag{5}$$

$$\mathcal{LB}_{ijl} = \mathcal{M}_t + \max_{i' \in \{1..n\}} [\sum_{o_{i'k} \in \mathcal{U}} \min_{l' \in \{1..m\}} (d_{i'kl'})] \tag{6}$$

Algorithm 1 Biased Policy Playout for ABGNRPA

Input: state, policy, heuristic_values, γ, τ
Begin
list_of_moves ← state.possible_moves() ▷ *Moves are tuples (job, operation, machine)*
while list_of_moves is not empty **do**
  for move ∈ list_of_moves **do**
    probabilities[move] ← $e^{\frac{\alpha_{move}}{\tau}+\beta_{move}}$ ▷ *table of* p_{move} *values* of size n_{moves}
  end for
  next_move ← `random_weighted_choice`(list_of_moves, probabilities)
  $\mathcal{LB}$ ← `LB_function`(next_move)
  state.play(next_move)
  for move ∈ list_of_moves **do**
    if move = next_move **then**
      $\delta \leftarrow 1$
    else
      $\delta \leftarrow \frac{-1}{\text{size(list\_of\_moves)}}$
    end if
    $\beta_{move} \leftarrow \beta_{move} + \gamma \ * \delta * \mathcal{LB}$
  end for
  list_of_moves ← state.possible_moves()
end while
End

We highlight that our contribution differs from [47]. Indeed, for the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), at each state they take into account the distance to the arrival point, the wait time and the arrival time, making the bias *dynamic*. In our case, we initialize the biases with the heuristic value and we update at each step them using $\mathcal{LB}$ which is computed from the current state instead of recomputing the bias. This means that the bias accumulates modifications every time the action is encountered.

4 Experiments and results

4.1 Model description and implementation

We construct our model considering the problem as a sequential decision making process by ordering the different operations on the machines using the inherent partial order given by the jobs, and compute the starting time of each of these operations after the allocation. The idea is to visualize the problem as a tree search problem and implement MCTS methods to solve it as explained in the following:

1. Since operations in a job have a specific order, we will allocate them following that order and we start by the first operation. The choices for the first action are then the couples containing the first operations of all jobs, *i.e*,

the subset $\mathcal{RO} = \{o_{i,0}\ ,\ i \in \{1..n\}\}$, and their compatible machines $\mathcal{A} = \{(o_{i,0}, m_l),,\ i \in \{1..n\}, l \in \{1..m\},$ if $o_{i,0}$ and m_l are compatible $\}$.
2. An operation is chosen, using the policy, and allocated on a compatible machine m_{l_0}. The starting time is then $t_{start,i_0,l_0} = 0$ and the ending time is $t_{end,i_0,0} = d_{i_0,0,m_{l_0}}$.
3. Given that we decided to process the operations following the job order, the only operation from job i_0 that can be allocated at this point is $o_{i_0,1}$. It then replaces $o_{i_0,0}$ in $\mathcal{RO}$.
4. We repeat the previous steps, each time removing the chosen operation and replacing it by the following one in the same job until there are no operations left in $\mathcal{RO}$.

We implemented the model using `Python`, version 3.9.19. All compared algorithms have been implemented from scratch and use native `Python` functions and class types, which enables a fair comparison of all baseline algorithms described in section 4.3.

4.2 Datasets

Several benchmarking instances are available in the literature [20] [3]. We choose to consider a subset of these instances, with $|\mathcal{M}_{ij}|$ representing the flexibility, *i.e* the number of compatible machines with operation o_{ij}:

- 10 instances from Brandimarte's [6]: $n = 10..20$, $m = 4..15$, $|\mathcal{J}_i| = 3..15$, $d_{ijl} = 1..20$, $|\mathcal{M}_{ij}| = 2..6$.
- 4 instances from Kacem's [28]: $n = 4..10$, $m = 5..10$, $\sum_i |\mathcal{J}_i| = 12..56$, $|\mathcal{M}_{ij}| = m$, $d_{ijl} = 1..10$ with some outliers such as 54 in *k1*
- 66x3 instances from Hurink's [26] (*edata, rdata, vdata*) derived from *sdata-instances*. With $n = 6..30$, $m = 5..15$, flexibility varies as:
  - *edata*: average is 1.15, max is 3 (few operations have flexibility)
  - *rdata*: average is 2, max is 3 (most operations have flexibility)
  - *vdata*: average is $\frac{1}{2}m$, max is $\frac{4}{5}m$ (all operations have flexibility).

All these descriptions are taken from [4]. These instances vary in size and flexibility, giving a variety of instance types to benchmark our approach. In addition, optimal values were found for all instances, except *mk10* from [6].

4.3 Baseline algorithms

We compare our algorithm to some variations of MCTS algorithms: classic MCTS, Nested Monte Carlo Tree Search (NMCTS) [8], Nested Rollout Policy Adaptation (NRPA) [14], Generalized Nested Rollout Policy Adaptation (GNRPA) [9] as well as other devised ones:

- *Heuristic-Based NMCTS* (**HBNMCTS**): the playouts are biased instead of uniformly random.

[3] `https://github.com/SchedulingLab/fjsp-instances`

- *Bias Initialized NRPA* (**BINRPA**): the weights of actions are initialized using the biases provided by the heuristic instead of a uniform distribution.
- *Randomized Greedy* (**RandGreedy**): no exploration is done, only a weighted random playout following the values of a heuristic.

All these algorithms are run for level 1, as explained in section 2.3 for GNRPA-based approaches and as described in [8] for NMCTS-based approaches. *Randomized Greedy* is the only exception as it only plays one rollout and does not conduct an exploration in the tree. We allow a budget of 60 seconds for every algorithm, running on 100 independent iterations.

4.4 Experimental Results

Algorithm	Best value frequency	UB reached count	Average STD	Gap to UB	
				Min makespan	Mean makespan
NRPA	1	1	7.59%	54.21%	83.16%
MCTS	2	2	2.88%	52.93%	66.66%
NMCTS	0	0	5.01%	61.00%	83.00%
HBNMCTS	32	3	3.05%	15.52%	23.91%
BINRPA	1	1	2.94%	52.71%	66.88%
RandGreedy	1	1	9.61%	53.33%	110.45%
GNRPA	72	4	2.71%	14.27%	18.79%
ABGNRPA	**126**	**8**	**1.43%**	**12.08%**	**17.84%**

Table 1: The table on the left showcases the number of times each algorithm obtains the best makespan compared to the others, and the number of times it finds the known Upper Bounds (UB) from literature [20,24]. The higher the value the better. The table on the right shows the average STandard Deviation (STD) of the makespan for each algorithm and the average gap on all instances between the upper bound and minimal and mean makespan values found. In this table, the lower the better.

Instanes groups	*edata*	*rdata*	*vdata*	Kacem	Brandimarte
Max deviation	31.51%	32.14 %	34.14%	28.06%	30.07%
Min deviation	7.59%	7.20%	5.30%	13.38%	10.19%
Average deviation	15.27%	14.59%	13.28%	20.86%	17.31%

Table 2: Bias deviation between the start and the end of an ABGNRPA run in percentages, aggregated on instances' datasets from [26,28,6]

An aggregation of the results obtained are available in table 1. Exhaustive results are shown in the appendix as per-instance results (minimal and average makespans and standard deviation) with averages ranking, all in section C. We also provide further details on nesting levels' impact in section B.4.

4.5 Observations and discussion

In table 1, we observe that, on **126 out of 212** instances, ABGNRPA performs better than the other algorithms. It gets closest to the optimal and even reaches it on **8 instances**, and GNRPA and HBNMCTS are right behind it. This shows that the addition of biases helps to improve the initial performance of NRPA. However, GNRPA does not appear to sufficiently exploit the information that can be extracted from the problem structure. The better performance of ABGNRPA implies that the adaptive bias feature abstracts additional information helping it get a better bias estimation. This is also highlighted by its average performance and stability. We can conjecture that the biases are *evolving* to become more compatible with the problem at hand and that this evolution, which occurs during the playouts, helps capture information before its end and capitalize on it to improve the search.

We compute the bias deviation between the start and the end of a ABGNRPA run. Aggregated results are in table 2, and per-instance results are available in section C.4 of the appendix. Varying between instances, this behavior can explain the better performance of ABGNRPA and indicates that the adaptation of the biases leads to a better representation of the instance's characteristics. However, contrary to other existing applications of GNRPA [19,12], increasing the nesting level does not improve performance for GNRPA variants. Our hypothesis is that these algorithms, with this specific model, stagnate around a local minimum.

The last point suggests that further training is rather useless as it would make the policy more deterministic. This hinders the possibility of further exploration beyond the visited branches. To counter this, we can consider using Limited Repetitions [11], a combination of different heuristics with a learning scheme for the weights on these heuristics from which the bias is constructed [48], or a nested bias system. Another possible improvement is the consideration of a state-action-based policy, and not only an action-based policy, which may resemble a Q-learning scheme [52]. For the $\mathcal{LB}$ function, we have tested the formulation mentioned in equation 6 but we can also try others that may be more expressive.

5 Conclusion

Through this work, we compared the performance of our algorithm, the ABGNRPA, to that of other MCTS variations on the FJSSP and concluded that we have indeed improved the results. However, many parameters are to be considered, such as the heuristic function, the $\mathcal{LB}$ estimation, and the hyper-parameters of the algorithm. The bias deviation, presented in table 2, seems to characterize the GNRPA family adaptability. This suggests the need for further investigation to fully explore its potential to handle larger and different problems. Improving the scope and extent of ABGNRPA may prove useful when tackling other planning and scheduling applications. Taking into account different factors that hinder the prowess of CP-based approaches, we can foresee the relevance of our approach in bypassing these obstacles with its stochastic nature.

References

1. Abgaryan, H., Harutyunyan, A., Cazenave, T.: Randomized greedy sampling for jssp. In: Proceedings of LION 18, the 18th Learning and Intelligent Optimization Conference. pp. 9–19 (2024)
2. Amiri, F., Shirazi, B., Tajdin, A.: Multi-objective simulation optimization for uncertain resource assignment and job sequence in automated flexible job shop. Applied Soft Computing **75**, 190–202 (2019). https://doi.org/10.1016/j.asoc.2018.11.015
3. Beck, J.C., Feng, T., Watson, J.P.: Combining constraint programming and local search for job-shop scheduling. INFORMS Journal on Computing **23**(1), 1–14 (2011)
4. Behnke, D., Geiger, M.J.: Test instances for the flexible job shop scheduling problem with work centers. Arbeitspapier/Research Paper/Helmut-Schmidt-Universität, Lehrstuhl für Betriebswirtschaftslehre, insbes. Logistik-Management (2012)
5. Bellman, R.: A markovian decision process. Journal of mathematics and mechanics pp. 679–684 (1957)
6. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations research **41**(3), 157–183 (1993)
7. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012)
8. Cazenave, T.: Nested monte-carlo search. In: Twenty-First International Joint Conference on Artificial Intelligence (2009)
9. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Carlo Search: First Workshop, MCS 2020, Held in Conjunction with IJCAI 2020, Virtual Event, January 7, 2021, Proceedings 1. pp. 71–83. Springer (2021)
10. Cazenave, T.: Monte carlo game solver. In: Monte Carlo Search: First Workshop, MCS 2020, Held in Conjunction with IJCAI 2020, Virtual Event, January 7, 2021, Proceedings 1. pp. 56–70. Springer (2021)
11. Cazenave, T.: Generalized nested rollout policy adaptation with limited repetitions. arXiv preprint arXiv:2401.10420 (2024). https://doi.org/10.48550/arXiv.2401.10420
12. Cazenave, T.: Learning a prior for monte carlo search by replaying solutions to combinatorial problems. In: Parallel Problem Solving from Nature, PPSN 2024. Lecture Notes in Computer Science, vol. 15148, pp. 85–99. Springer (2024), https://doi.org/10.1007/978-3-031-70055-2_6
13. Cazenave, T., Lucas, J.Y., Kim, H., Triboulet, T.: Monte carlo vehicle routing. In: ATT at ECAI 2020 (2020)
14. Cazenave, T., Saffidine, A., Schofield, M., Thielscher, M.: Nested monte carlo search for two-player games. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)
15. Chang, J., Yu, D., Hu, Y., He, W., Yu, H.: Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. Processes **10**(4), 760 (2022). https://doi.org/10.3390/pr10040760
16. Chaudhry, I.A., Khan, A.A.: A research survey: review of flexible job shop scheduling techniques. International Transactions in Operational Research **23**(3), 551–591 (2016)

17. Clark, G.: Deep synoptic monte-carlo planning in reconnaissance blind chess. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 4106–4119. Curran Associates, Inc. (2021), https://proceedings.neurips.cc/paper_files/paper/2021/file/215a71a12769b056c3c32e7299f1c5ed-Paper.pdf
18. Da Col, G., Teppan, E.C.: Industrial-size job shop scheduling with constraint programming. Operations Research Perspectives **9**, 100249 (2022)
19. Dang, C., Bazgan, C., Cazenave, T., Chopin, M., Wuillemin, P.: Warm-starting nested rollout policy adaptation with optimal stopping. In: Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023. pp. 12381–12389. AAAI Press (2023), https://doi.org/10.1609/aaai.v37i10.26459
20. Dauzère-Pérès, S., Ding, J., Shen, L., Tamssaouet, K.: The flexible job shop scheduling problem: A review. European Journal of Operational Research **314**(2), 409–432 (2024)
21. De Giovanni, L., Pezzella, F.: An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. European journal of operational research **200**(2), 395–408 (2010)
22. Eliahou, S., Fonlupt, C., Fromentin, J., Marion-Poty, V., Robilliard, D., Teytaud, F.: Investigating monte-carlo methods on the weak schur problem. In: European Conference on Evolutionary Computation in Combinatorial Optimization. pp. 191–201. Springer (2013)
23. Fera, M., Fruggiero, F., Lambiase, A., Martino, G., Nenni, M.E., et al.: Production scheduling approaches for operations management. InTech (2013)
24. des Graviers, M.E.C., Kobrosly, L., Guettier, C., Cazenave, T.: Updating lower and upper bounds for the job-shop scheduling problem test instances (2025). https://doi.org/10.48550/arXiv.2504.16106, https://arxiv.org/abs/2504.16106
25. Hameed, M.S.A., Schwung, A.: Reinforcement learning on job shop scheduling problems using graph networks. arXiv preprint arXiv:2009.03836 (2020)
26. Hurink, J., Jurisch, B., Thole, M.: Tabu search for the job-shop scheduling problem with multi-purpose machines. Operations-Research-Spektrum **15**, 205–215 (1994)
27. Jamrus, T., Chien, C.F., Gen, M., Sethanan, K.: Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing. IEEE Transactions on Semiconductor Manufacturing **31**(1), 32–41 (2017). https://doi.org/10.1109/TSM.2017.2758380
28. Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Mathematics and computers in simulation **60**(3-5), 245–276 (2002)
29. Karimi, S., Ardalan, Z., Naderi, B., Mohammadi, M.: Scheduling flexible job-shops with transportation times: Mathematical models and a hybrid imperialist competitive algorithm. Applied mathematical modellíng **41**, 667–682 (2017). https://doi.org/10.1016/j.apm.2016.09.022
30. Kasapidis, G.A., Paraskevopoulos, D.C., Repoussis, P.P., Tarantilis, C.D.: Flexible job shop scheduling problems with arbitrary precedence graphs. Production and Operations Management **30**(11), 4044–4068 (2021)
31. Kemmerling, M., Lütticke, D., Schmitt, R.H.: Beyond games: a systematic review of neural monte carlo tree search applications. Applied Intelligence **54**(1), 1020–1046 (2024). https://doi.org/10.1007/s10489-023-05240-w
32. Li, K., Deng, Q., Zhang, L., Fan, Q., Gong, G., Ding, S.: An effective mcts-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. Computers & Industrial Engineering **155**, 107211 (2021)

33. Li, Y.: Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274 (2017)
34. Liu, C.L., Chang, C.C., Tseng, C.J.: Actor-critic deep reinforcement learning for solving job shop scheduling problems. Ieee Access **8**, 71752–71762 (2020)
35. Lu, C.L., Chiu, S.Y., Wu, J., Chao, L.P.: Dynamic monte-carlo tree search algorithm for multi-objective flexible job-shop scheduling problem. Appl. Math **10**(4), 1531–1539 (2016)
36. Luo, Q., Deng, Q., Gong, G., Zhang, L., Han, W., Li, K.: An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. Expert Systems with Applications **160**, 113721 (2020)
37. Mastrolilli, M., Svensson, O.: Hardness of approximating flow and job shop scheduling problems. J. ACM **58**(5) (Oct 2011). https://doi.org/10.1145/2027216.2027218
38. Meilanitasari, P., Shin, S.J.: A review of prediction and optimization for sequence-driven scheduling in job shop flexible manufacturing systems. Processes **9**(8), 1391 (2021). https://doi.org/10.3390/pr9081391
39. Meng, L., Zhang, C., Ren, Y., Zhang, B., Lv, C.: Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. Computers & industrial engineering **142**, 106347 (2020)
40. Naderi, B., Roshanaei, V.: Critical-path-search logic-based benders decomposition approaches for flexible job shop scheduling. INFORMS Journal on Optimization **4**(1), 1–28 (2022)
41. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling **8**, 145–159 (2005)
42. Park, J., Chun, J., Kim, S.H., Kim, Y., Park, J.: Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. International Journal of Production Research **59**(11), 3360–3377 (2021)
43. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Computers & operations research **35**(10), 3202–3212 (2008)
44. Rajabinasab, A., Mansour, S.: Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach. The International Journal of Advanced Manufacturing Technology **54**, 1091–1107 (2011). https://doi.org/10.1007/s00170-010-2986-7
45. Rosin, C.D.: Nested rollout policy adaptation for monte carlo tree search. In: Ijcai. vol. 2011, pp. 649–654 (2011)
46. Saqlain, M., Ali, S., Lee, J.: A monte-carlo tree search algorithm for the flexible job-shop scheduling in manufacturing systems. Flexible Services and Manufacturing Journal **35**(2), 548–571 (2023)
47. Sentuc, J., Cazenave, T., Lucas, J.Y.: Generalized nested rollout policy adaptation with dynamic bias for vehicle routing. arXiv preprint arXiv:2111.06928 (2021)
48. Sentuc, J., Ellouze, F., Lucas, J.Y., Cazenave, T.: Learning the bias weights for generalized nested rollout policy adaptation. In: International Conference on Learning and Intelligent Optimization. pp. 194–207. Springer (2023). https://doi.org/10.1007/978-3-031-44505-7_14
49. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)

50. Song, W., Chen, X., Li, Q., Cao, Z.: Flexible job-shop scheduling via graph neural network and deep reinforcement learning. IEEE Transactions on Industrial Informatics **19**(2), 1600–1610 (2022)
51. Vodopivec, T., Samothrakis, S., Ster, B.: On monte carlo tree search and reinforcement learning. Journal of Artificial Intelligence Research **60**, 881–936 (2017)
52. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698
53. Watson, J.P., Beck, J., Howe, A.E., Whitley, L.: Problem difficulty for tabu search in job-shop scheduling. Artificial Intelligence **143**(2), 189–217 (2003). https://doi.org/10.1016/S0004-3702(02)00363-6
54. Wu, T.Y., Wu, I.C., Liang, C.C.: Multi-objective flexible job shop scheduling problem based on monte-carlo tree search. In: 2013 Conference on Technologies and Applications of Artificial Intelligence. pp. 73–78. IEEE (2013)
55. Xie, J., Gao, L., Peng, K., Li, X., Li, H.: Review on flexible job shop scheduling. IET collaborative intelligent manufacturing **1**(3), 67–77 (2019)
56. Yang, X., Yoshizoe, K., Taneda, A., Tsuda, K.: Rna inverse folding using monte carlo tree search. BMC bioinformatics **18**, 1–12 (2017). https://doi.org/10.1186/s12859-017-1882-7
57. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Chi, X.: Learning to dispatch for job shop scheduling via deep reinforcement learning. Advances in neural information processing systems **33**, 1621–1632 (2020)
58. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling. ieee transactions on cybernetics **51**(4), 1797–1811 (2020)
59. Zhou, J.: A constraint program for solving the job-shop problem. In: Principles and Practice of Constraint Programming—CP96: Second International Conference, CP96 Cambridge, MA, USA, August 19–22, 1996 Proceedings 2. pp. 510–524. Springer (1996)
60. Zhou, L., Zhang, L., Horn, B.K.: Deep reinforcement learning-based dynamic scheduling in smart manufacturing. Procedia Cirp **93**, 383–388 (2020)

Appendix

A Relevance study

A.1 Description

This section aims at studying the relevance of using MCTS and its variants on the Flexible Job Shop Scheduling Problem. The intuition is, as proposed in [22], that the average makespan of random playouts in a branch is expected to provide a hint on how promising this branch is to explore. Other metrics could be explored but are not considered in this study. In theory, if we consider X the random variable representing the makespan of a playout, a good estimator of the expected value $\mathbf{E}(X)$ is the mean of the playouts $\bar{X}$. We devide the search tree into distinct branches, where in our case, a branch of the search tree starts by assigning sequentially two operations on machines and then all the possible outcomes after these first two steps. Then, we fully explore each branch and compute the average and minimal makespans for this given branch. We then evaluate the correlation between the mean and minimum given by all branches to assess whether the mean makespan is a good indicator of the branch's ability to provide a good solution. Studying the standard deviation did not provide substantial information except the fact that the deviation of the larger makespans compared to the mean is greater than that of the smaller ones.

A.2 Test instances

We consider a small subset of the instances described in section 4.2 as well as others available in the literature [4]. For each group of datasets, we choose a random instance and reduce its content to explore the search tree fully and reasonably quickly. Table 3 summarizes the size of each instance considered.

Reduced instance	n	m	n_i	Flexibility	$d_{j,i}$
mt10xxx	4	13	2..3	1..3	10..98
lar03_1	4	60	2..3	1..2	10..30
mk03	4	8	2..3	2..4	1..19
05a	3	5	4..5	1..3	18..84
mfjs04	4	7	2..3	2..3	65..247
abz9	3	15	3..5	2..3	11..40
k3	4	10	1..3	1..7	1..10

Table 3: Description of the instances considered for the relevance study

A.3 Experimental results

The optimal values and correlation between minimal and average makespans computed for each branch are shown in table 4. An example of how the mean and minimal makespan vary in each branch is shown in figure 3.

Reduced Instance	**mt10xxx**	**lar03_1**	**mk03**	**05a**	**mfjs04**	**abz9**	**k3**
Correlation	0.643	0.685	0.709	0.438	0.770	0.749	0.887
Optimal	197	67	21	305	496	170	7
Number of branches	32	60	111	42	68	56	194

Table 4: Correlation values, optimal values and number of branches for the instances described in table 3

.

We notice that on most instances, the correlation factor is relatively high, which is a good indicator of the validity of using MCTS methods on this type of problems. The low correlation value for the simplified instance of *05a* is a result of the fact that we can find a near-optimal solution in a high number of branches, while the mean value of the makespan differs between branches.

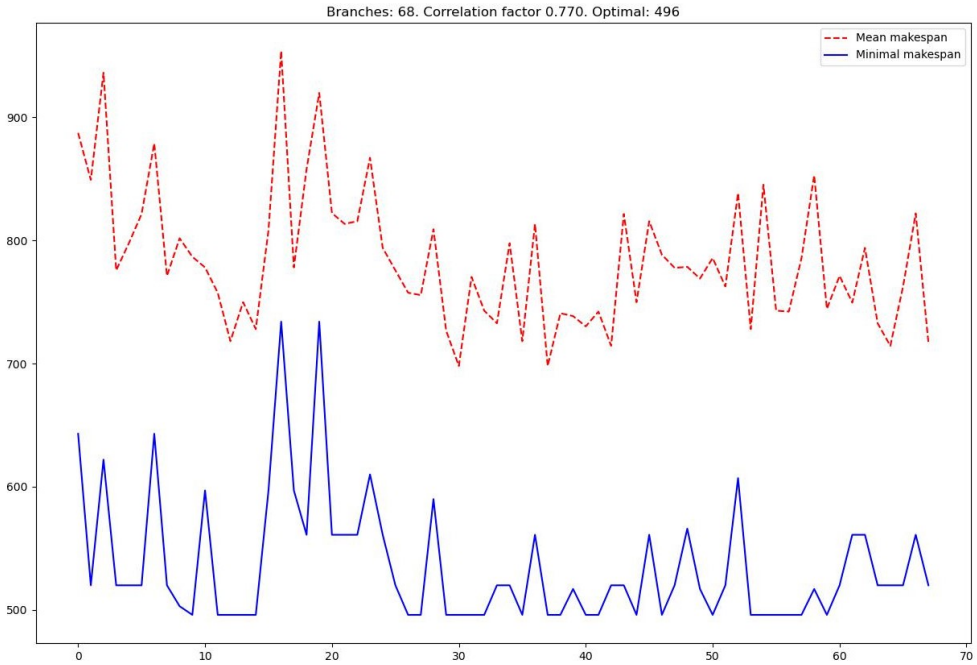

Fig. 3: Correlation between mean and minimal makespans for each branch for the reduced mfjs04 instance.

B Parameters of the experimental procedure

B.1 The choice of the heuristic for bias computation

When using GNRPA as a baseline to compare our algorithm's performance, we faced the task of choosing the appropriate heuristic value for computing the biases. We came up thus with a list of choices, some of them were described in the literature as usable for JSSP instances and others were of our own creation:

- *Shortest Processing Time* (**SPT**): *i.e*, the shorter the processing time of the combination of (operation, machine), the higher the bias value
- *Longest Processing Time* (**LPT)**): we prioritize the couples (operation, machine) with the longest processing time.
- *Least Remaining Processing Time* (**LRPT**): we prioritize the operation of the job that has the least remaining time (we consider the lowest processing time for each of the remaining operations for this job).
- *Most Remaining Processing Time* (**MRPT**): the prioritization process is the opposite of the last heuristic.
- *Earliest Ending Time* (**EET**): we prioritize the operations that would have the earliest ending time. For that, we compute the feasible start time with the addition of the processing time of the machine of the action, given the current state.
- *Latest Ending Time* (**LET**): operations with the latest ending time are prioritized.

We compare the performance of the GNRPA for each of these heuristics on some of the instances in the body of the article under the same conditions. The results shown in table 5 show that the SPT and EET heuristics perform better than the pessimistic ones. In particular, EET provides the best results. This shows how important the choice of the heuristic is. When we use the same comparison for ABGNRPA, we observe similar behavior. These empirical results are the basis of the use of this heuristic in our experimental procedure.

B.2 ABGNRPA performance with different heuristics

It seems essential to validate the compatibility between our algorithm and the heuristics we chose to test. While the EET heuristic provides the best results for the GNRPA as well as the ABGNRPA, as mentioned above, the better performance of the ABGNRPA compared to the other algorithms does not necessarily hold true when using a different heuristic, as shown in table 6.

We observe that the improvement in performance holds for the SPT heuristic in the same way as for EET, while being unfavorable for the other ones, especially when it comes to pessimistic heuristics.

B.3 Correlation between the bias values and the policy values

For both GNRPA and ABGNRPA, we observe better performance in general compared to the *Randomized Greedy* baseline. This justifies the thought that adding a policy to the biases improves performance. We then decide to verify the correlation between the policy weights and the biases at the end of the running of each algorithm, and we obtained similar results for ABGNRPA and GNRPA. The **average correlation is 0.2**, which means that the heuristic serves as a good start for these algorithms but they learn to deviate from the initial weights. In the same sense, this also justifies the better performance of ABGNRPA relatively to that of GNRPA as it shifts the biases as well.

Instance	LRPT	MRPT	SPT	LPT	EET	LET
k1	13	14	13	15	**12**	14
k2	29	29	29	30	**10**	28
k3	21	23	24	26	**8**	20
k4	46	47	46	43	**12**	47
mk01	61	61	62	61	**44**	61
mk02	59	59	59	58	**35**	59
mk03	336	333	343	349	**222**	338
mk04	105	100	97	109	**71**	101
mk05.	239	234	237	238	**184**	238
mk06	174	176	167	180	**70**	172
mk07	262	258	255	246	**169**	250
mk08	664	653	655	657	**525**	658
mk09	588	571	600	592	**384**	590
mk10	490	513	516	524	**305**	515

Table 5: Comparison of the performance of GNRPA (minimal makespan on 100 runs of the algorithm) using different heuristics

Instance	MRPT		LRPT		LPT		SPT	
	GNRPA	ABGNRPA	GNRPA	ABGNRPA	GNRPA	ABGNRPA	GNRPA	ABGNRPA
k1	14	67	13	12	15	66	13	11
k2	29	191	29	15	30	172	29	14
k3	23	83	21	9	26	81	24	8
k4	47	180	46	26	43	223	46	16
mk01	61	76	61	84	61	75	62	51
mk02	59	75	59	80	58	84	59	38
mk03	333	491	336	585	349	412	343	233
mk04	100	140	105	216	109	132	97	101
mk05	234	218	239	492	238	226	237	210
mk06	176	220	174	272	180	235	167	100
mk07	258	315	262	360	246	313	255	171
mk08	653	679	664	1866	657	645	655	571
mk09	571	680	588	1554	592	518	600	418
mk10	513	599	490	1262	524	455	516	343

Table 6: Comparison of GNRPA and ABGNRPA using different heuristics

B.4 The effect of increasing nesting levels

	ABGNRPA		GNRPA	
Instance	Level 1	Level 2	Level 1	Level 2
k1	11	11	12	12
k2	11	11	13	13
k3	7	7	8	8
k4	12	12	12	12
mk01	44	43	45	44
mk02	32	32	34	35
mk03	226	232	220	222
mk04	67	67	70	71
mk05	181	181	183	184
mk06	66	65	69	70
mk07	164	164	165	169
mk08	527	529	528	525
mk09	347	351	386	384
mk10	279	278	295	305

Table 7: Comparison of minimal makespan obtained by GNRPA-based algorithms with one or two levels of nesting

We observe in table 7 that increasing the level does not improve the performance of GNRPA and ABGNRPA. This is not the case for NRPA and NMCTS as shown in table 8.

	NMCTS		NRPA	
Instance	Level 1	Level 2	Level 1	Level 2
k1	11	11	12	11
k2	25	18	25	16
k3	18	14	21	13
k4	32	29	38	28

Table 8: Comparison of minimal makespan obtained with NMCTS and NRPA with nesting of levels 1 and 2

B.5 The use of 2-opt permutations

Since we are constructing a sequence through the resolution of our problem, we figured it may be interesting to try `2-opt` permutations on this sequence at the end of a playout to see if we can improve the makespan through local search. In short, these permutations do improve the performance of all algorithms except ABGNRPA, but do not exceed its performance, which strengthens the idea that we are saturating the performance of the chosen tree search methods on this problem.

C Detailed performance metrics

C.1 Per-instance minimal makespan

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA	Upper bound
abz5	1534	1503	1671	1218	1471	1506	**1212**	**1212**	1167*
abz6	1149	1172	1274	1011	1198	1237	997	**978**	925*
abz7	1191	1078	1128	754	1083	1075	778	**742**	610
abz8	1262	1138	1210	785	1176	1127	799	**776**	636
abz9	1217	1104	1161	821	1123	1114	837	**808**	644*
car1	7127	7980	8469	7422	7544	7416	**6708**	6726	6176*
car2	7462	7881	8161	7212	7766	7534	**6534**	6586	6327*
car3	7918	8404	8624	7902	8584	8107	**7190**	7400	6856*
car4	8568	8975	9369	8591	9110	8586	**7997**	8033	7789*
car5	7835	8219	8536	7656	8368	7767	**7591**	**7591**	7229*
car6	9323	9622	9946	8584	9710	9151	**8173**	8570	7990*
car7	6671	6579	7385	6539	6972	6510	**6123**	**6123**	6132*
car8	8699	9024	9664	7985	9031	8437	8006	8089	7689*
la01	694	709	682	**639**	707	749	658	645	609*
la02	733	744	703	708	744	779	**692**	**692**	655*
la03	640	638	659	609	624	628	606	**586**	550*
la04	640	671	689	629	655	698	638	**615**	568*
la05	555	566	595	573	560	604	529	**522**	503*
la06	940	961	1009	912	989	1011	903	**857**	833*
la07	928	943	976	857	942	931	817	**808**	762*
la08	951	921	1042	943	978	983	922	**890**	845*
la09	1015	1023	1087	942	1023	1016	914	**896**	878*
la10	982	996	1049	985	994	1027	952	**892**	866*
la11	1308	1286	1422	1224	1303	1308	1196	**1185**	1103*
la12	1149	1159	1177	1082	1123	1165	1025	**982**	960*
la13	1297	1241	1327	1207	1243	1250	1134	**1095**	1053*
la14	1276	1325	1411	1282	1311	1335	1202	**1168**	1123*
la15	1421	1388	1440	1241	1402	1397	**1203**	1235	1111*
la16	1195	1168	1231	986	1150	1191	973	**964**	892*
la17	973	887	1042	771	956	954	**775**	**775**	707*
la18	1044	1074	1100	928	1058	1077	**910**	913	842*
la19	1059	1069	1141	842	1048	1084	810	**806**	796*
la20	1093	1163	1220	956	1116	1139	**898**	913	857*
la21	1616	1493	1560	1187	1490	1500	1198	**1191**	1009*
la22	1277	1344	1462	1081	1358	1297	**1052**	1063	880*
la23	1457	1478	1529	1097	1434	1445	1076	**1073**	950*
la24	1388	1399	1511	1112	1384	1421	**1064**	1069	908*
la25	1475	1412	1495	1146	1399	1430	**1091**	1102	936*
la26	1885	1856	1868	1455	1858	1806	1362	1391	1106*
la27	1975	1893	1878	1532	1923	1919	**1445**	1459	1181*
la28	1964	1810	1962	1456	1885	1870	**1397**	1412	1142*
la29	1909	1784	1875	1457	1759	1872	**1382**	1401	1107*
la30	2065	1905	2049	1552	1935	1977	**1496**	1507	1188*
la31	2655	2505	2583	1910	2586	2409	1837	**1830**	1532*
la32	2983	2680	2758	2145	2686	2649	2033	**1944**	1698*
la33	2533	2393	2501	1928	2414	2384	1842	**1782**	1574*
la34	2628	2565	2619	1954	2594	2496	1919	**1850**	1599*
la35	2927	2608	2762	2049	2600	2631	**2004**	2005	1736*
la36	2123	1871	2037	**1360**	1896	1838	1401	1424	1160*
la37	2172	2061	2211	1573	2073	2049	**1524**	1552	1397*
la38	2003	1912	2025	1396	1818	1878	**1297**	1351	1141*
la39	2059	1905	2048	1401	1961	1880	1382	**1371**	1184*
la40	2012	1862	1992	1290	1905	1816	1267	**1261**	1144*
mt06	**55***	**55***	61	57	**55***	**55***	60	**55***	55*
mt10	1139	1214	1251	1018	1183	1220	969	**966**	871*
mt20	1443	1433	1504	1275	1470	1482	**1204**	1212	1088*
orb1	1309	1329	1466	1124	1349	1351	**1065**	1074	977*
orb10	1240	1189	1328	1042	1226	1258	**1030**	1038	933*
orb2	1119	1123	1188	965	1077	1127	935	**930**	865*
orb3	1300	1312	1426	1080	1293	1273	**1032**	1042	951*
orb4	1272	1326	1418	1098	1265	1325	**1104**	1112	984*
orb5	1100	1120	1171	953	1144	1119	928	**913**	842*
orb6	1271	1288	1492	1041	1331	1330	1029	**1019**	958*
orb7	515	489	537	422	517	525	**416**	418	389*
orb8	1150	1126	1256	958	1137	1166	**923**	940	894*
orb9	1210	1205	1264	**1022**	1182	1273	1034	1026	933*

Table 9: Experimental results on Hurink's [26] *edata*, with a time budget of 60 seconds. * means the value is optimal.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA	Upper bound
abz5	1498	1544	1713	1127	1588	1557	**1039**	1040	954*
abz6	1238	1168	1227	906	1096	1205	**857**	878	807*
abz7	1237	1137	1137	684	1134	1134	759	**651**	522
abz8	1282	1171	1218	693	1156	1121	761	**684**	535
abz9	1254	1166	1181	717	1137	1137	780	**714**	536
car1	6218	6590	6949	5940	6494	6077	**5400**	5455	5034*
car2	6751	7240	7257	6551	6979	6820	6226	**6190**	5985*
car3	7114	7136	7880	6410	7222	6728	**6195**	6375	5622*
car4	7503	7616	8043	7277	8012	7411	6973	**6897**	6514*
car5	6819	7429	8218	6801	7357	7237	**6227**	6426	5615*
car6	8085	8529	8761	6858	8511	7523	**6659**	6816	6147*
car7	6009	6048	6282	5299	6029	5447	**4744**	4855	4425*
car8	7682	7802	7889	6598	7677	7328	6148	**6143**	5692*
la01	702	722	779	637	729	769	616	**604**	570*
la02	643	655	708	591	657	693	604	**595**	529*
la03	547	597	657	560	617	613	**516**	534	477*
la04	619	669	676	571	649	659	**567**	**567**	502*
la05	541	555	623	514	548	575	506	**505**	457*
la06	931	999	1043	871	967	982	**864**	868	799*
la07	904	912	1000	840	931	900	800	**798**	749*
la08	914	923	984	848	958	975	**825**	831	765*
la09	1038	1048	1134	919	1032	1051	896	**893**	853*
la10	917	1002	1062	911	997	995	875	**867**	804*
la11	1203	1347	1426	1205	1345	1361	**1157**	1161	1071*
la12	1108	1096	1204	1058	1174	1166	1009	**978**	936*
la13	1235	1274	1268	1158	1241	1284	1117	**1094**	1038*
la14	1240	1248	1397	1173	1317	1358	1135	**1126**	1070*
la15	1311	1354	1414	1193	1336	1321	1160	**1136**	1089*
la16	1124	1149	1240	831	1113	1160	805	**794**	717*
la17	970	978	942	728	927	995	710	**709**	646*
la18	1005	1067	1135	732	984	1082	743	**724**	666*
la19	1067	1116	1159	818	1052	1119	792	**784**	700*
la20	1193	1154	1274	879	1157	1155	**876**	878	756*
la21	1516	1555	1632	**993**	1530	1469	1038	1010	825
la22	1380	1384	1485	1006	1312	1413	999	**965**	753
la23	1540	1507	1636	1070	1543	1567	1049	**1028**	831
la24	1512	1464	1409	1078	1434	1479	1047	**1029**	795
la25	1533	1431	1534	1062	1347	1482	**1022**	1041	779
la26	2015	1876	2017	**1315**	1841	1868	1343	1327	1057
la27	1956	2002	2035	1379	1909	1792	1385	**1357**	1085*
la28	1922	1921	2077	1381	1938	1895	**1344**	1351	1076
la29	1859	1820	1871	1334	1861	1765	**1280**	1313	994
la30	2135	1926	2016	**1389**	1925	1839	1407	**1389**	1071
la31	2652	2419	2656	1885	2593	2429	1878	**1750**	1520*
la32	2973	2819	2893	2018	2814	2733	2079	**1895**	1657*
la33	2740	2616	2596	1833	2613	2562	1832	**1730**	1497*
la34	2787	2603	2632	1903	2581	2497	1911	**1779**	1535*
la35	2775	2621	2635	1857	2594	2554	1919	**1855**	1549*
la36	2058	2004	1998	1313	1949	1981	1286	**1260**	1023*
la37	2306	1991	2213	1300	2078	1996	1351	**1250**	1062*
la38	2176	1899	2088	**1220**	1873	1834	1245	1242	954*
la39	2021	1969	2034	1252	1928	1838	1276	**1239**	1011*
la40	2045	1947	1858	1201	1977	1965	1182	**1167**	955*
mt06	52	53	52	53	53	57	**49**	**49**	47*
mt10	1059	1067	1135	819	1096	1081	**787**	**787**	686*
mt20	1363	1342	1416	1158	1273	1304	**1114**	1135	1022*
orb1	1071	1165	1264	903	1140	1175	**876**	883	746*
orb10	1175	1162	1226	889	1156	1209	865	**854**	742*
orb2	1078	1061	1198	833	1051	1098	**787**	793	696*
orb3	1114	1168	1245	850	1088	1174	819	**817**	712*
orb4	1195	1190	1338	880	1176	1178	870	**856**	753*
orb5	1011	1074	1146	791	1050	1071	**760**	767	639*
orb6	1197	1165	1293	874	1176	1214	868	**840**	754*
orb7	417	488	514	342	489	522	336	**333**	302*
orb8	974	1002	1089	733	1018	1038	718	**706**	639*
orb9	1105	1039	1204	823	1082	1157	805	**797**	694*

Table 10: Experimental results on Hurink's [26] *rdata*, with a time budget of 60 seconds. * means the value is optimal.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA	Upper bound
abz5	1473	1485	1610	899	1495	1493	972	**893**	859*
abz6	1195	1177	1246	786	1207	1226	808	**772**	742*
abz7	1188	1117	1114	612	1111	1090	645	**598**	492*
abz8	1218	1136	1083	**610**	1145	1167	712	**610**	507
abz9	1216	1122	1106	**621**	1088	1126	700	622	497*
car1	6248	6294	6830	5528	6135	5689	5460	**5384**	5005*
car2	6564	6657	7016	6201	6829	6360	**6084**	6284	5929*
car3	6821	7370	7591	6322	7070	6717	**6108**	6236	5597*
car4	6970	7600	7700	6964	7440	7185	6819	**6804**	6514*
car5	6923	6921	7431	5843	7142	6325	**5626**	5691	4909*
car6	7787	7536	8758	5532	7998	6559	5781	**5486**	5486*
car7	5651	5884	6037	4598	5782	5065	4592	**4381**	4281*
car8	6943	6982	7872	5188	7085	6336	5471	**4987**	4613*
la01	674	716	756	607	719	723	596	**590**	570*
la02	646	650	699	570	660	680	585	**579**	529*
la03	562	606	637	553	607	619	**526**	530	477*
la04	602	597	632	561	617	640	553	**549**	502*
la05	562	577	617	505	581	613	**500**	505	457*
la06	924	1017	1069	862	999	1013	864	**880**	799*
la07	919	918	1001	832	935	946	**797**	819	749*
la08	928	961	966	859	930	927	**801**	844	765*
la09	955	1021	1063	901	1054	1049	**898**	905	853*
la10	905	960	996	875	992	969	**857**	878	804*
la11	1279	1296	1374	1144	1347	1339	**1103**	1143	1071*
la12	1165	1162	1179	1037	1140	1174	**981**	1008	936*
la13	1274	1274	1291	1136	1282	1277	**1101**	1102	1038*
la14	1234	1289	1390	1178	1308	1325	1126	**1118**	1070*
la15	1282	1338	1395	1151	1316	1341	**1132**	1146	1089*
la16	1044	1093	1180	**734**	1044	1038	760	739	717*
la17	947	921	1003	**651**	952	962	659	663	646*
la18	1051	967	1132	671	1038	1059	707	**663**	663*
la19	1016	1097	1138	698	1033	1007	700	**685**	617*
la20	1073	1008	1072	771	1078	1110	760	**758**	756*
la21	1473	1464	1521	**977**	1483	1499	980	981	800*
la22	1384	1348	1455	918	1368	1275	928	**909**	733*
la23	1552	1520	1602	976	1440	1490	1027	**970**	809*
la24	1499	1452	1537	**959**	1452	1416	993	980	773*
la25	1343	1402	1481	**958**	1464	1441	971	960	751*
la26	1966	1874	1906	1294	1908	1833	1350	**1289**	1084*
la27	2111	1917	1913	**1287**	1944	1883	1370	1305	1069*
la28	2020	1962	2004	1319	1888	1872	1363	**1318**	993*
la29	1846	1858	1905	**1225**	1809	1783	1252	1230	1068*
la30	2025	1941	1902	1335	1896	1891	**1319**	1320	1068*
la31	2825	2592	2626	1778	2615	2465	1932	**1690**	1520*
la32	2872	2730	2686	1964	2758	2676	2114	**1918**	1657*
la33	2719	2511	2526	1744	2545	2490	1875	**1700**	1497*
la34	2811	2489	2640	1838	2582	2442	1952	**1736**	1535*
la35	2772	2592	2503	1823	2626	2389	1956	**1769**	1549*
la36	2126	1865	1969	1043	1931	1870	1119	**1033**	948*
la37	2172	2096	2007	**1112**	2059	1886	1178	1124	986*
la38	1965	1852	1936	**1055**	1906	1804	1170	1074	943*
la39	1861	1936	1891	**1039**	1848	1867	1103	1045	922*
la40	1979	1828	1865	1043	1884	1846	1172	**1030**	955*
mt06	51	48	51	**47***	49	55	**47***	**47***	47*
mt10	1031	1028	1111	701	1041	1059	711	**682**	655*
mt20	1203	1256	1293	1119	1269	1282	1072	**1069**	1022*
orb1	1147	1126	1159	766	1140	1143	753	**735**	695*
orb10	1116	1083	1201	753	1058	1126	754	**720**	681*
orb2	1058	1052	1124	**700**	1053	1109	719	703	620*
orb3	1028	1088	1107	**692**	1058	1047	713	693	648*
orb4	1100	1081	1238	**757**	1072	1102	785	761	753*
orb5	952	985	1069	643	951	1011	638	**623**	584*
orb6	1040	1098	1151	733	1154	1113	737	**723**	715*
orb7	468	478	491	321	477	507	319	**315**	275*
orb8	940	892	1039	627	930	944	614	**587**	573*
orb9	928	1105	1120	706	989	1097	720	**696**	659*

Table 11: Experimental results on Hurink's [26] *vdata*, with a time budget of 60 seconds. * means the value is optimal [20,24].

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA	Upper bound
k1	12	**11***	12	**11***	12	12	**11***	**11***	11*
k2	24	25	27	13	25	28	**11***	**11***	11*
k3	18	22	24	7	20	22	**7***	**7***	7*
k4	37	39	42	13	41	46	**12**	**12**	11*
mk01	55	58	61	44	57	62	**43**	44	40*
mk02	48	51	59	34	50	55	32	**31**	26*
mk03	330	338	352	238	331	339	222	**213**	204*
mk04	95	94	106	72	93	98	70	**69**	60*
mk05	214	225	237	188	222	231	187	**183**	172*
mk06	170	163	164	71	164	177	**68**	**68**	57*
mk07	236	251	255	170	241	254	163	**158**	139*
mk08	701	651	675	547	663	662	526	**523***	523*
mk09	614	595	580	358	559	582	390	**356**	307*
mk10	496	509	503	303	506	504	310	**267**	193

Table 12: Experimental results on Kacem's [28] and Brandimarte's [6] instances minimum makespan on 100 iterations of each algorithm with a time budget of 60 seconds for each iteration. * means the value is optimal.

C.2 Per-instance average makespan

NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
6.43	4.40	6.70	2.78	4.55	7.86	1.68	**1.61**

Table 13: Average ranking of tested algorithms on the mean makespan obtained on 100 iterations with a budget of 60 seconds on Hurink's, Kacem's, and Brandimarte's instances

Instance	NRPA	MCTS	nmcts	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	1835.42	1628.19	1841.72	1309.56	1625.47	2079.37	1247.22	1244.25
abz6	1436.91	1283.75	1489.44	1081.84	1295.42	1657.60	1025.44	1027.38
abz7	1355.30	1156.27	1217.89	793.33	1156.75	1365.68	803.09	774.38
abz8	1433.11	1219.66	1307.34	831.05	1227.09	1440.15	841.94	804.92
abz9	1375.52	1184.63	1269.44	858.33	1193.77	1399.35	858.84	849.19
car1	8774.19	8551.98	9690.52	8423.45	8497.53	10135.55	6965.39	6990.89
car2	8690.52	8317.16	9270.20	7832.48	8321.70	9738.73	6742.86	6822.25
car3	9554.25	9330.59	10409.92	8817.38	9286.11	10724.66	7402.56	7714.63
car4	9769.02	9600.03	10565.33	9294.36	9623.08	10836.40	8221.95	8368.23
car5	9244.89	8945.91	10197.78	8267.27	8964.97	10395.17	7610.25	7638.14
car6	10535.83	10445.16	11971.06	9419.97	10511.39	11607.40	8474.70	8774.31
car7	7934.45	7428.78	8953.70	7797.06	7443.41	8940.06	6171.27	6273.50
car8	9823.39	9500.23	10830.34	9068.20	9485.84	11028.66	8294.02	8398.06
la01	814.67	756.19	870.69	739.52	759.53	1034.73	658.17	658.16
la02	832.17	788.80	874.16	775.95	787.36	1039.09	710.80	709.91
la03	733.61	672.77	765.95	655.72	675.66	916.28	613.84	608.03
la04	780.75	722.17	811.09	702.97	718.30	994.01	649.55	645.77
la05	669.47	618.47	717.56	620.75	622.28	876.97	555.47	559.44
la06	1131.92	1035.63	1165.42	972.41	1032.67	1352.71	939.02	956.30
la07	1076.22	1014.59	1125.72	904.78	1007.81	1307.54	835.14	835.67
la08	1117.55	1042.92	1162.20	1001.47	1037.89	1355.89	946.03	966.70
la09	1193.34	1086.30	1223.58	994.91	1094.92	1421.25	932.33	930.80
la10	1133.86	1051.81	1174.58	1051.16	1046.09	1368.46	984.11	997.00
la11	1501.69	1401.09	1548.47	1278.81	1396.81	1746.18	1233.08	1255.70
la12	1313.20	1234.22	1352.83	1125.84	1232.56	1558.88	1065.55	1090.84
la13	1448.84	1333.69	1451.78	1252.08	1327.98	1664.91	1168.63	1199.13
la14	1509.91	1398.58	1528.41	1362.73	1405.42	1738.35	1246.19	1318.66
la15	1576.78	1486.86	1609.58	1323.28	1486.92	1827.36	1251.78	1274.23
la16	1394.25	1247.91	1400.14	1047.28	1250.25	1594.52	994.20	997.92
la17	1178.77	1032.22	1213.27	851.09	1036.17	1366.11	795.17	802.86
la18	1243.22	1146.77	1308.95	976.11	1143.00	1483.66	922.20	927.80
la19	1261.38	1146.02	1322.02	944.70	1147.55	1486.33	831.55	831.23
la20	1338.95	1228.83	1382.97	1064.56	1228.95	1576.30	942.91	957.28
la21	1844.59	1622.05	1799.30	1300.13	1630.78	1998.18	1246.47	1262.64
la22	1635.95	1466.78	1596.45	1161.13	1455.11	1812.35	1100.47	1124.70
la23	1767.94	1568.22	1726.58	1170.66	1566.69	1931.11	1115.20	1131.72
la24	1724.58	1510.81	1665.64	1209.95	1509.78	1864.61	1114.06	1126.80
la25	1740.39	1546.00	1719.36	1277.91	1550.61	1935.06	1155.83	1163.03
la26	2232.02	1970.98	2116.41	1527.05	1971.17	2333.59	1444.03	1468.67
la27	2310.59	2037.94	2190.19	1616.89	2025.13	2402.10	1496.27	1554.13
la28	2282.89	1992.28	2146.97	1509.48	1992.92	2367.17	1450.91	1480.52
la29	2272.70	1959.30	2138.86	1519.05	1965.22	2325.02	1440.89	1474.94
la30	2369.31	2070.98	2249.02	1631.52	2082.94	2471.11	1546.50	1600.03
la31	3114.56	2691.20	2829.30	1973.33	2711.47	3045.67	1939.94	1951.45
la32	3312.91	2855.97	2995.36	2234.48	2869.38	3235.54	2108.84	2185.17
la33	3061.77	2607.20	2757.70	2002.80	2637.23	2977.61	1899.06	1981.41
la34	3132.95	2722.61	2847.73	2081.14	2718.94	3064.95	1981.09	2037.92
la35	3288.63	2827.02	2963.50	2166.14	2851.53	3212.81	2086.63	2116.20
la36	2410.06	2057.98	2238.17	1550.97	2057.11	2436.02	1474.22	1487.64
la37	2546.69	2216.69	2407.61	1661.03	2208.61	2583.25	1582.55	1596.45
la38	2360.05	2035.38	2234.89	1483.17	2028.17	2405.37	1378.69	1417.08
la39	2387.00	2062.13	2260.73	1480.94	2078.58	2452.77	1427.25	1428.13
la40	2395.23	2021.25	2206.81	1370.03	2026.83	2389.35	1331.27	1313.84
mt06	61.77	56.20	78.81	70.77	56.72	82.93	60.95	57.89
mt10	1406.64	1302.00	1460.08	1106.94	1294.42	1633.75	1010.20	1017.50
mt20	1623.94	1528.98	1667.72	1348.45	1545.84	1879.00	1233.28	1256.61
orb1	1522.77	1418.06	1603.17	1188.42	1430.98	1773.35	1097.23	1099.11
orb10	1431.16	1313.22	1499.67	1129.30	1313.58	1666.63	1060.41	1058.75
orb2	1350.53	1228.78	1401.52	1058.33	1226.34	1579.39	966.58	978.39
orb3	1521.13	1411.47	1585.31	1166.89	1411.58	1774.52	1060.63	1076.70
orb4	1545.23	1400.67	1601.41	1179.22	1398.52	1762.72	1143.31	1156.64
orb5	1272.66	1190.91	1344.25	1021.86	1199.83	1506.09	956.09	958.11
orb6	1559.23	1430.27	1630.42	1152.73	1436.55	1802.82	1050.84	1052.77
orb7	606.95	549.83	626.53	450.95	551.25	721.05	434.11	429.56
orb8	1333.11	1238.84	1391.70	1041.94	1244.17	1580.45	976.44	978.50
orb9	1444.81	1319.52	1501.22	1121.91	1315.28	1681.18	1050.70	1059.44

Table 14: Experimental results on Hurink's [26] *edata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	1891.25	1712.56	1956.19	1206.38	1716.08	2189.99	1113.81	1101.56
abz6	1461.58	1308.70	1494.52	970.53	1311.58	1708.67	914.59	913.16
abz7	1423.41	1217.55	1271.08	716.66	1217.02	1439.06	791.03	694.67
abz8	1489.42	1257.86	1318.66	741.81	1257.48	1487.41	814.02	725.08
abz9	1433.48	1225.86	1286.73	752.52	1225.78	1443.50	814.02	737.73
car1	7328.19	7212.58	8056.66	6281.98	7165.17	8186.89	5627.53	5688.83
car2	7919.41	7620.02	8366.36	6890.48	7574.45	8751.56	6426.31	6495.05
car3	8256.55	7897.73	9025.81	6926.91	7966.20	8816.94	6383.94	6560.31
car4	8843.92	8492.84	9541.75	7774.02	8525.80	9757.88	7202.94	7211.50
car5	8458.81	8156.88	9369.94	7437.83	8117.48	9152.65	6578.44	6776.75
car6	10121.53	9488.44	10693.77	7637.39	9513.67	10113.75	6996.48	7097.63
car7	6999.50	6518.61	7446.70	6072.19	6534.06	7769.51	5037.23	5068.03
car8	8832.45	8443.94	9437.94	7169.52	8360.16	9368.84	6421.61	6330.23
la01	828.63	774.48	882.59	696.13	780.56	1077.65	637.17	634.03
la02	754.73	710.78	810.06	656.67	708.89	990.59	625.94	630.25
la03	682.77	653.19	754.63	610.92	657.64	934.07	552.77	555.95
la04	738.13	704.22	806.45	635.25	705.17	999.80	584.91	588.00
la05	655.23	613.23	711.70	568.38	611.58	892.09	535.88	536.17
la06	1101.48	1061.13	1193.97	922.38	1056.41	1389.91	885.56	898.03
la07	1072.81	1010.39	1126.33	875.34	1009.28	1326.61	829.61	834.30
la08	1075.17	1021.67	1144.11	918.61	1032.45	1355.27	855.73	881.03
la09	1206.09	1123.05	1249.50	962.38	1125.72	1452.68	925.92	934.55
la10	1117.09	1080.63	1195.13	965.45	1072.52	1413.75	907.30	936.22
la11	1515.73	1436.98	1571.20	1257.42	1435.95	1794.79	1190.23	1219.52
la12	1328.88	1235.56	1336.50	1089.23	1240.13	1558.94	1041.70	1061.52
la13	1400.70	1354.03	1451.73	1193.97	1347.83	1686.09	1142.98	1168.69
la14	1460.89	1404.63	1527.92	1224.08	1409.41	1761.38	1161.11	1191.48
la15	1511.72	1444.27	1547.83	1245.36	1438.02	1782.04	1183.45	1212.80
la16	1387.33	1250.31	1415.80	903.88	1246.81	1626.57	832.92	823.34
la17	1145.02	1048.16	1186.88	811.42	1054.44	1385.28	732.50	738.19
la18	1252.48	1147.66	1311.80	841.30	1149.64	1519.62	788.72	771.13
la19	1308.17	1186.61	1356.97	896.77	1184.75	1548.74	832.75	829.28
la20	1408.17	1259.55	1427.08	959.13	1256.20	1640.73	902.73	905.52
la21	1850.34	1654.91	1807.48	1103.31	1651.59	2026.25	1086.52	1067.94
la22	1729.45	1520.00	1628.56	1081.98	1517.20	1877.82	1032.53	1035.00
la23	1935.36	1669.44	1845.78	1118.11	1677.11	2058.12	1094.17	1063.81
la24	1779.44	1576.47	1710.50	1143.22	1579.97	1949.62	1086.36	1090.41
la25	1806.89	1565.23	1695.89	1125.69	1555.27	1938.53	1072.30	1079.95
la26	2371.86	2044.50	2194.52	1391.89	2059.56	2420.36	1379.08	1366.33
la27	2375.14	2110.52	2250.11	1446.17	2114.03	2493.52	1434.45	1414.77
la28	2427.67	2115.61	2256.11	1541.42	2114.52	2496.81	1440.59	1488.08
la29	2287.31	1967.38	2093.77	1406.44	1977.88	2324.99	1333.25	1359.55
la30	2437.28	2118.88	2269.72	1506.83	2129.67	2504.01	1452.75	1455.69
la31	3189.14	2750.45	2828.61	1930.34	2753.08	3089.40	1934.78	1909.92
la32	3449.08	3015.97	3103.00	2092.44	3014.78	3372.86	2130.81	2071.28
la33	3257.81	2783.08	2899.91	1909.52	2802.02	3130.41	1911.61	1886.13
la34	3241.39	2782.38	2850.42	1957.30	2774.59	3102.16	1955.89	1933.94
la35	3273.86	2835.33	2905.94	1924.17	2858.44	3182.33	1983.25	1914.81
la36	2477.11	2156.27	2309.13	1404.33	2160.66	2536.96	1385.58	1339.41
la37	2635.73	2253.06	2386.47	1374.39	2244.52	2626.32	1410.39	1327.58
la38	2427.69	2077.73	2236.25	1321.00	2074.22	2449.54	1304.28	1285.64
la39	2463.38	2107.77	2272.16	1321.16	2104.50	2472.90	1327.81	1278.39
la40	2431.94	2113.63	2256.41	1256.05	2107.58	2489.63	1266.72	1210.38
mt06	61.94	56.95	77.19	67.61	57.02	86.34	49.00	49.02
mt10	1289.31	1178.73	1305.30	878.19	1178.64	1523.09	824.16	824.44
mt20	1490.63	1430.25	1546.58	1224.23	1431.47	1775.38	1150.36	1183.27
orb1	1351.05	1281.17	1429.48	984.98	1263.42	1633.10	916.53	918.56
orb10	1389.20	1289.63	1453.47	943.78	1290.52	1657.82	892.80	883.23
orb2	1283.53	1180.11	1333.31	877.19	1183.23	1534.31	832.84	827.38
orb3	1361.09	1263.78	1430.67	908.27	1266.64	1623.89	852.80	851.70
orb4	1466.00	1299.03	1468.59	942.38	1302.77	1703.74	895.44	886.47
orb5	1228.19	1152.77	1289.25	839.38	1147.78	1487.34	796.50	793.16
orb6	1428.47	1296.19	1469.72	951.31	1296.53	1674.66	895.28	891.92
orb7	568.84	533.47	598.84	375.66	529.97	710.81	356.67	348.50
orb8	1215.45	1120.67	1250.78	795.25	1114.61	1454.65	744.03	738.28
orb9	1323.56	1209.13	1369.84	885.09	1202.89	1579.72	845.38	836.36

Table 15: Experimental results on Hurink's [26] *rdata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	1753.33	1664.14	1790.33	942.42	1655.78	2093.46	1017.55	918.59
abz6	1374.70	1294.81	1406.89	821.77	1298.84	1657.11	851.88	791.39
abz7	1355.19	1188.02	1205.69	629.69	1192.33	1397.60	663.17	634.59
abz8	1434.38	1244.05	1252.42	630.97	1245.67	1471.03	738.08	634.88
abz9	1367.28	1195.95	1215.84	647.36	1189.09	1404.20	728.50	648.89
car1	7302.58	6861.69	7815.27	5845.20	6858.05	7871.34	5641.80	5682.13
car2	7618.09	7322.77	8152.55	6636.67	7349.72	8397.75	6324.17	6465.84
car3	8136.58	7908.70	8863.75	6774.98	7903.28	8766.58	6296.73	6503.94
car4	8201.83	8124.88	9118.95	7348.33	8096.58	9120.40	6980.25	7039.48
car5	8028.72	7787.84	8720.78	6229.30	7806.06	8410.53	5938.33	5932.19
car6	9166.13	8602.98	9754.89	5943.58	8650.72	9025.97	6151.16	5614.20
car7	6818.94	6364.75	7453.11	5117.36	6461.42	7475.17	4802.11	4487.47
car8	8170.03	7707.33	8788.58	5640.98	7751.95	8233.49	5762.44	5206.55
la01	804.69	766.88	873.27	642.50	765.14	1059.45	615.44	611.42
la02	740.30	708.09	816.13	648.72	708.48	997.89	603.45	604.39
la03	678.69	650.77	740.16	595.17	651.00	909.23	549.44	557.31
la04	675.38	647.89	737.20	602.59	652.72	900.39	574.73	579.91
la05	662.02	632.73	718.48	562.72	628.75	896.38	522.67	531.08
la06	1118.89	1067.81	1179.11	936.78	1070.50	1396.56	898.42	924.45
la07	1028.27	998.89	1102.44	874.20	1008.11	1313.61	820.41	841.39
la08	1070.28	1028.70	1152.84	901.89	1034.28	1355.46	840.41	875.33
la09	1135.16	1117.09	1222.84	951.19	1123.44	1457.40	915.03	924.72
la10	1086.67	1047.00	1162.23	945.39	1059.88	1367.99	884.20	915.75
la11	1458.66	1399.27	1495.03	1190.48	1413.53	1754.44	1152.63	1173.97
la12	1276.81	1240.11	1329.34	1066.58	1236.89	1566.35	1014.17	1039.83
la13	1410.52	1373.63	1465.08	1171.81	1372.73	1719.61	1124.61	1148.97
la14	1456.25	1397.53	1515.72	1232.25	1399.30	1750.68	1151.13	1203.00
la15	1504.70	1446.36	1558.89	1201.44	1453.41	1808.16	1163.97	1183.13
la16	1299.13	1214.61	1302.38	790.13	1207.73	1554.95	788.94	758.56
la17	1098.06	1050.73	1147.47	709.72	1047.77	1374.89	684.25	688.16
la18	1201.13	1137.81	1251.23	699.81	1146.08	1482.70	726.48	680.66
la19	1217.27	1169.52	1263.75	737.23	1168.97	1498.74	720.56	710.41
la20	1239.78	1176.05	1279.28	821.06	1192.09	1516.84	780.41	786.88
la21	1790.58	1624.91	1722.66	1032.44	1624.92	1970.80	1020.44	1013.14
la22	1648.05	1496.95	1579.75	953.98	1500.61	1842.34	965.47	936.69
la23	1808.95	1638.25	1725.91	1039.66	1627.33	1994.37	1075.59	1010.48
la24	1699.36	1552.27	1647.41	1034.41	1561.72	1885.37	1039.56	1020.66
la25	1679.86	1545.97	1632.88	1020.39	1549.83	1903.17	1026.58	999.34
la26	2286.70	2045.39	2114.58	1347.05	2055.34	2420.63	1411.27	1343.94
la27	2372.19	2091.47	2165.83	1371.14	2105.69	2443.11	1440.66	1365.52
la28	2320.22	2072.55	2161.38	1373.27	2064.23	2410.04	1428.42	1363.94
la29	2239.50	1981.42	2051.34	1285.47	1984.58	2342.08	1318.98	1270.25
la30	2359.44	2084.77	2174.20	1388.36	2096.91	2448.84	1363.03	1371.16
la31	3209.94	2776.16	2830.08	1839.13	2789.64	3115.88	2040.70	1847.75
la32	3370.47	2975.92	2987.17	2010.30	2969.00	3254.27	2211.42	2016.91
la33	3114.67	2701.61	2744.36	1799.38	2717.16	3021.27	1978.59	1807.73
la34	3139.36	2756.14	2805.86	1888.88	2761.69	3055.43	2051.19	1892.38
la35	3197.03	2777.89	2834.77	1871.41	2799.50	3088.79	2064.98	1877.28
la36	2345.64	2063.06	2134.13	1093.02	2073.33	2395.62	1147.16	1087.64
la37	2500.34	2238.52	2297.94	1154.00	2231.97	2576.62	1231.64	1155.88
la38	2299.34	2044.77	2103.53	1117.59	2043.55	2356.53	1209.36	1118.33
la39	2335.91	2075.22	2128.89	1121.09	2080.42	2405.43	1151.41	1118.22
la40	2308.91	2034.11	2107.61	1078.84	2033.77	2355.17	1218.45	1071.52
mt06	59.36	55.36	62.70	53.22	55.39	85.16	47.00	47.00
mt10	1215.70	1140.69	1256.28	751.09	1137.59	1485.81	746.39	718.50
mt20	1374.63	1346.95	1434.30	1174.31	1345.30	1663.02	1104.42	1156.13
orb1	1291.33	1219.72	1329.06	788.02	1219.36	1572.06	798.13	761.50
orb10	1299.73	1216.64	1343.44	787.61	1223.59	1581.67	791.20	751.00
orb2	1252.13	1158.89	1281.05	760.67	1170.73	1505.15	755.91	728.77
orb3	1243.95	1193.73	1312.20	735.11	1186.66	1565.41	751.39	710.42
orb4	1356.89	1265.81	1412.41	803.39	1276.78	1652.74	837.22	787.23
orb5	1128.41	1086.31	1170.06	670.41	1080.42	1394.91	661.36	649.84
orb6	1283.13	1224.94	1324.89	757.81	1228.80	1560.03	759.84	736.59
orb7	556.80	524.70	572.75	354.72	520.44	702.75	331.70	338.45
orb8	1097.69	1044.73	1159.91	655.11	1054.52	1408.92	645.56	627.80
orb9	1267.41	1195.69	1300.83	747.06	1183.64	1558.15	760.22	728.19

Table 16: Experimental results on Hurink's [26] *vdata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
mk01	67.00	63.17	75.41	50.09	62.45	96.52	45.11	45.66
mk02	59.33	57.14	65.88	39.70	57.48	86.07	34.63	36.09
mk03	408.11	359.81	389.72	269.34	359.36	476.90	231.94	258.23
mk04	114.84	106.78	122.22	77.66	106.66	150.34	72.48	72.72
mk05	253.03	238.27	261.22	197.73	239.86	302.56	190.30	190.27
mk06	208.17	179.52	194.03	74.02	179.11	237.95	71.61	70.78
mk07	283.17	269.80	287.25	184.92	271.03	364.70	173.97	178.97
mk08	787.59	695.94	740.86	569.97	702.75	830.21	537.14	554.47
mk09	721.95	628.06	654.89	390.09	625.98	766.02	405.45	380.50
mk10	625.13	537.20	560.25	340.69	537.80	665.88	322.20	333.16
k1	17.02	13.05	21.89	12.45	13.20	34.18	11.00	11.00
k2	32.50	28.91	34.42	14.61	29.13	69.72	12.00	11.97
k3	26.73	24.53	28.64	8.11	24.61	48.41	7.19	7.03
k4	47.72	45.98	51.16	14.27	46.66	65.07	12.59	13.47

Table 17: Experimental results on Kacem's [28] and Brandimarte's [6] instances: average makespan on 100 iterations of each algorithm with a time budget of 60 seconds for each iteration.

C.3 Standard deviation of the makespan

NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
6.89	4.08	5.94	3.30	4.17	7.93	2.09	**1.61**

Table 18: Average ranking of tested algorithms on the standard deviation of the makespan obtained on 100 iterations with a budget of 60 seconds on Hurink's, Kacem's, and Brandimarte's instances

Instance	NRPA	MCTS	nmcts	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	139.44	47.50	100.15	55.17	50.87	195.08	18.90	14.11
abz6	133.32	37.59	80.93	32.38	31.34	152.45	96.49	9.73
abz7	76.62	29.51	44.19	18.31	27.66	99.57	12.78	13.37
abz8	100.14	32.20	51.68	20.64	24.35	100.25	18.14	11.33
abz9	78.65	30.18	38.45	15.09	27.02	96.85	17.38	12.97
car1	720.27	266.85	870.88	417.85	313.23	895.63	13.29	98.80
car2	707.89	196.23	573.26	360.23	223.54	790.31	14.95	88.50
car3	719.11	310.82	939.65	527.00	295.23	948.69	11.93	104.30
car4	579.22	260.31	658.34	317.30	225.64	767.42	95.51	135.36
car5	579.54	295.80	824.99	400.37	245.12	919.63	19.41	17.53
car6	719.60	293.12	801.15	439.86	293.20	951.49	27.68	84.25
car7	527.36	208.59	798.00	624.26	192.40	784.00	19.67	43.11
car8	698.28	273.03	817.64	604.48	238.47	890.78	8.13	71.28
la01	59.17	16.14	64.74	42.78	17.08	114.50	18.29	2.60
la02	47.59	17.49	59.72	34.71	17.88	100.68	83.21	6.34
la03	61.85	15.39	55.09	26.10	18.32	97.46	8.28	3.46
la04	71.08	19.04	62.38	39.11	20.97	109.29	5.64	8.21
la05	53.76	19.39	62.19	27.27	18.12	104.77	35.42	6.58
la06	88.31	24.50	67.26	31.61	24.02	128.20	93.46	14.31
la07	73.87	26.93	48.01	26.52	27.69	119.86	7.90	8.90
la08	87.79	26.57	56.09	23.95	24.41	131.04	9.12	14.35
la09	97.71	26.53	64.13	24.99	26.90	135.46	14.19	8.71
la10	87.68	23.20	50.76	27.60	21.73	132.51	9.82	12.85
la11	99.38	34.95	57.48	25.66	39.64	143.12	13.50	14.04
la12	85.28	26.11	55.86	24.31	34.87	139.47	21.05	11.04
la13	99.63	27.42	62.01	23.89	35.17	143.45	15.75	13.54
la14	99.42	27.58	48.01	35.63	37.25	143.64	18.25	21.60
la15	95.45	36.64	67.01	31.16	34.12	144.55	10.44	14.17
la16	114.26	35.04	73.27	28.67	33.04	142.62	76.53	12.60
la17	114.52	37.96	63.43	41.15	29.35	135.45	13.43	9.96
la18	102.03	27.64	76.49	24.11	33.36	144.10	16.82	5.27
la19	96.82	30.73	78.73	42.56	33.75	144.28	1.36	14.76
la20	102.52	31.42	71.14	46.20	34.91	152.03	17.44	19.78
la21	161.21	47.19	86.02	47.90	44.59	170.29	5.42	21.92
la22	147.79	40.75	74.22	32.56	39.67	157.07	9.92	19.31
la23	153.21	41.80	83.12	30.19	44.67	165.23	96.04	18.70
la24	149.96	48.59	73.89	43.39	44.69	165.00	21.18	23.36
la25	131.66	51.68	88.78	56.97	51.71	168.57	24.86	25.25
la26	166.03	42.83	91.54	33.56	44.16	185.96	16.73	25.46
la27	145.58	59.92	100.55	39.87	53.41	180.33	10.71	28.19
la28	174.27	60.54	80.16	26.49	50.01	179.23	24.33	19.22
la29	182.88	58.19	84.22	29.43	58.78	173.24	7.98	18.06
la30	157.85	58.37	90.66	39.93	56.28	186.93	18.32	31.28
la31	172.06	62.11	104.48	31.51	56.93	195.70	16.81	32.99
la32	197.62	67.43	96.67	44.79	66.86	204.13	28.80	33.27
la33	215.72	76.69	83.95	35.97	68.57	200.29	9.89	34.66
la34	227.93	66.64	90.38	36.14	58.00	195.13	10.79	26.87
la35	180.42	73.95	79.39	37.24	63.54	202.85	11.53	26.74
la36	190.24	61.13	94.98	55.01	59.35	185.87	4.68	28.31
la37	178.83	59.19	100.57	39.50	51.36	188.78	13.11	13.53
la38	194.15	50.88	99.86	34.92	65.03	185.85	28.49	27.14
la39	174.49	56.48	89.28	36.33	48.82	192.63	9.80	18.88
la40	199.47	58.24	90.85	30.76	48.35	184.75	11.95	18.36
mt06	4.71	0.89	10.88	9.38	1.02	10.95	12.14	1.63
mt10	112.13	32.94	75.44	34.04	36.67	141.64	13.90	16.30
mt20	104.03	31.85	63.50	28.36	34.39	141.64	0.21	16.75
orb1	104.90	36.16	72.02	36.86	32.66	147.99	21.17	11.64
orb10	102.75	40.58	81.47	47.11	37.17	143.00	17.64	10.06
orb2	96.89	34.88	71.87	39.55	38.01	148.83	10.43	18.20
orb3	89.56	42.52	81.61	41.83	40.56	149.86	3.85	13.63
orb4	122.15	36.48	72.92	36.12	41.99	151.87	12.71	23.37
orb5	84.00	33.67	72.94	31.22	23.72	127.39	13.67	12.45
orb6	123.30	41.26	73.70	46.32	36.93	155.64	21.65	12.10
orb7	48.76	18.41	36.32	17.35	15.70	68.73	10.19	5.93
orb8	87.37	37.19	67.55	37.06	37.07	132.04	20.86	13.66
orb9	136.19	37.24	84.52	50.37	44.51	150.07	7.97	12.89

Table 19: Experimental results on Hurink's [26] *edata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	173.31	57.75	86.47	39.04	52.36	210.20	21.61	20.61
abz6	104.32	46.48	86.52	24.82	43.94	172.62	78.08	14.03
abz7	90.91	27.78	45.56	17.17	33.70	106.26	15.45	15.95
abz8	96.27	33.10	45.72	15.35	32.23	110.60	13.09	12.16
abz9	89.39	31.28	39.51	11.43	29.65	103.98	19.11	10.19
car1	527.04	202.97	498.77	231.49	223.59	706.92	10.39	66.65
car2	578.36	175.20	513.97	224.37	206.92	720.62	14.16	76.32
car3	638.89	227.24	507.16	261.85	205.54	718.52	10.30	92.14
car4	600.42	229.42	598.38	198.71	224.19	772.95	84.33	84.97
car5	612.87	213.79	543.00	276.97	247.86	752.43	115.34	105.56
car6	783.42	290.17	589.88	408.25	329.74	793.55	20.37	130.46
car7	608.79	186.11	575.25	281.24	185.52	775.92	19.12	85.68
car8	644.91	242.63	609.18	318.44	216.00	723.97	8.94	68.33
la01	62.93	19.26	50.69	32.22	21.64	121.76	25.96	7.39
la02	62.12	21.55	52.23	27.83	19.12	112.63	75.99	10.38
la03	59.07	21.67	55.77	22.06	16.09	110.75	15.21	8.07
la04	56.73	17.59	51.27	30.83	19.00	118.79	6.65	9.24
la05	65.36	18.35	52.52	29.63	21.72	112.20	80.36	9.63
la06	85.40	25.85	60.86	20.94	30.65	136.54	85.88	10.88
la07	86.90	28.54	44.76	20.74	26.34	130.98	9.67	10.85
la08	71.88	31.64	61.68	31.00	32.72	136.81	11.08	13.77
la09	102.18	26.30	49.78	16.92	26.34	138.36	13.33	9.77
la10	82.51	28.37	58.69	30.98	30.77	141.55	7.96	12.80
la11	101.97	33.28	55.63	25.11	38.57	154.43	10.37	16.75
la12	114.49	33.61	52.84	17.06	27.74	138.61	22.76	10.97
la13	94.85	32.76	55.24	19.62	39.80	146.03	18.26	9.94
la14	89.12	35.14	64.23	21.20	36.21	152.22	23.18	11.77
la15	100.06	37.96	53.56	19.75	39.32	148.95	11.82	12.71
la16	128.57	40.25	66.80	30.14	49.08	160.58	95.36	13.30
la17	102.39	30.98	69.54	44.26	34.20	142.43	13.77	10.81
la18	116.53	37.30	71.22	37.10	44.25	158.63	18.01	16.81
la19	108.97	34.28	67.94	25.50	36.75	154.75	7.40	14.68
la20	122.00	40.94	68.18	28.48	45.36	162.29	15.42	13.12
la21	159.66	44.59	67.83	28.80	47.01	176.46	15.22	15.78
la22	139.29	41.29	66.03	31.98	55.89	160.47	11.83	20.56
la23	158.32	54.18	80.40	25.84	54.46	178.15	122.94	15.19
la24	120.78	53.16	70.85	29.01	50.62	167.84	24.49	20.13
la25	144.78	49.60	74.21	27.05	56.74	174.05	26.56	15.84
la26	183.91	56.94	76.60	26.44	58.29	189.82	11.47	16.79
la27	193.63	49.38	79.12	30.13	61.03	195.56	10.57	21.87
la28	183.38	65.67	71.33	55.35	55.12	190.26	16.94	39.28
la29	175.89	61.63	83.74	26.73	52.19	180.03	10.10	19.43
la30	176.48	63.11	82.79	36.21	68.49	193.25	12.34	23.45
la31	205.80	76.95	76.73	20.26	69.50	207.06	19.93	19.79
la32	224.60	85.55	86.97	29.32	75.79	213.43	21.66	23.24
la33	199.38	73.75	87.72	33.11	81.52	201.05	12.56	23.40
la34	207.33	59.43	86.29	25.60	79.55	204.52	16.22	29.49
la35	224.63	65.39	93.54	26.06	76.41	208.75	18.43	19.95
la36	175.20	56.43	105.93	43.52	58.53	200.98	10.10	27.47
la37	202.88	69.38	79.18	26.81	63.50	203.91	12.23	19.61
la38	166.53	62.31	67.95	34.03	64.93	200.12	28.02	17.55
la39	175.76	51.74	92.44	33.49	60.41	196.61	10.04	16.02
la40	186.28	51.19	108.50	28.04	54.91	195.20	13.47	16.70
mt06	5.33	1.60	12.44	8.44	1.51	11.37	12.74	0.12
mt10	116.26	40.81	62.26	33.30	35.88	145.31	14.66	16.02
mt20	82.06	38.97	60.86	23.78	40.39	150.43	0.00	15.22
orb1	101.92	40.32	68.77	38.28	39.62	151.64	18.88	14.69
orb10	117.95	37.92	89.84	26.51	37.40	160.67	15.90	11.49
orb2	117.02	36.45	65.86	21.15	35.86	156.53	13.08	14.60
orb3	123.05	37.73	76.89	28.05	42.71	147.43	9.40	10.83
orb4	135.89	38.68	67.82	27.75	45.87	173.11	11.84	11.98
orb5	107.38	34.67	64.76	23.24	34.44	145.69	11.29	11.36
orb6	126.73	42.92	72.19	29.75	45.16	161.13	20.83	13.65
orb7	53.26	16.61	35.97	13.75	17.06	73.76	12.91	6.03
orb8	96.40	35.00	66.51	30.53	36.51	142.14	19.35	11.28
orb9	123.02	43.38	79.13	31.55	50.51	161.34	11.49	16.11

Table 20: Experimental results on Hurink's [26] *rdata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
abz5	130.86	48.01	89.13	15.92	58.34	205.11	19.69	10.03
abz6	98.09	39.59	71.95	16.33	43.57	163.30	83.48	11.05
abz7	71.12	33.91	32.27	10.18	33.90	104.98	13.83	12.94
abz8	75.24	33.89	38.38	8.05	37.82	115.79	10.00	9.21
abz9	71.88	30.09	32.20	9.47	37.76	105.15	17.26	9.72
car1	677.85	200.17	470.40	152.02	195.85	733.74	12.60	90.75
car2	555.98	193.94	401.64	195.55	206.95	716.50	12.38	73.86
car3	585.32	202.13	500.69	239.29	260.64	774.79	10.63	99.64
car4	545.76	204.48	524.05	190.17	250.19	691.47	81.02	80.98
car5	672.23	244.76	516.40	159.58	256.20	726.73	103.05	98.64
car6	836.04	333.89	537.95	155.55	281.28	868.05	30.78	72.81
car7	605.65	206.36	620.78	247.66	231.14	802.21	25.67	52.47
car8	697.66	245.65	496.07	160.71	208.71	720.79	7.58	80.99
la01	64.49	23.82	52.68	16.99	20.84	119.49	21.20	5.82
la02	48.21	21.89	54.48	23.55	22.24	116.72	73.10	9.38
la03	58.68	18.69	42.14	18.92	18.61	103.71	14.33	8.57
la04	48.68	17.78	45.90	19.96	17.33	99.81	5.05	7.93
la05	63.37	18.45	43.84	24.53	18.57	105.63	80.13	8.78
la06	93.43	21.32	53.89	37.26	28.07	139.05	80.86	14.09
la07	66.24	30.78	49.17	15.48	27.88	128.50	11.00	9.67
la08	79.45	32.99	46.08	18.90	27.98	133.38	14.77	11.40
la09	79.58	29.31	57.97	19.08	25.48	143.54	11.37	8.57
la10	83.64	30.92	54.54	20.21	23.94	133.51	7.94	9.11
la11	97.27	34.13	53.06	15.45	28.97	152.01	10.19	10.28
la12	78.10	31.94	57.11	14.38	33.65	140.56	19.33	10.46
la13	98.62	35.41	65.99	15.59	42.25	150.37	12.00	11.00
la14	100.29	33.72	54.43	22.71	32.61	150.61	16.31	13.65
la15	106.14	34.94	70.82	16.17	42.58	162.02	8.97	9.38
la16	113.61	29.90	53.35	17.50	41.21	156.71	114.16	9.99
la17	95.94	40.35	54.58	16.01	34.99	141.40	12.27	10.20
la18	81.74	40.89	55.54	13.53	36.28	147.99	17.53	6.96
la19	104.84	29.74	53.36	15.74	44.49	148.15	6.31	9.61
la20	76.65	41.70	57.38	21.65	40.00	150.45	13.76	12.80
la21	131.22	50.03	60.83	18.46	55.23	174.73	9.23	13.97
la22	125.76	47.86	55.70	15.19	47.87	170.49	10.63	9.66
la23	146.72	48.95	68.88	25.16	54.82	179.35	133.96	16.91
la24	108.68	47.96	54.02	22.62	40.39	166.63	32.46	15.96
la25	123.74	49.18	66.85	22.61	41.23	174.65	30.21	13.71
la26	135.48	59.41	76.09	21.22	56.22	206.27	13.05	18.67
la27	140.87	57.68	76.10	25.65	58.58	191.95	10.30	21.39
la28	154.40	52.85	54.26	23.37	58.65	188.46	21.65	22.77
la29	153.50	51.24	71.73	25.39	63.65	193.64	8.40	18.30
la30	157.78	57.83	82.69	24.56	61.38	198.46	15.86	21.53
la31	174.35	81.03	78.14	21.22	80.89	221.76	20.53	23.24
la32	183.88	73.09	81.97	19.05	70.98	212.76	34.87	28.34
la33	141.98	73.46	66.90	15.77	74.27	206.02	8.18	17.31
la34	168.26	67.29	71.79	21.14	76.42	199.91	12.30	19.70
la35	172.76	75.64	87.43	15.88	65.78	200.36	14.49	18.92
la36	98.59	60.71	67.69	17.10	64.81	189.33	6.87	16.21
la37	130.86	67.01	91.74	17.73	54.86	194.15	11.35	14.44
la38	135.06	64.70	67.77	20.19	60.24	187.15	13.61	19.63
la39	152.62	62.64	80.01	21.50	61.25	193.07	9.72	25.24
la40	138.26	59.51	72.13	16.67	66.14	184.19	13.94	15.77
mt06	5.17	1.80	7.49	4.15	2.00	11.39	11.69	0.00
mt10	92.62	34.23	55.36	19.95	37.13	150.90	12.73	12.93
mt20	95.06	28.37	44.31	23.49	33.07	144.19	0.00	12.56
orb1	89.66	38.17	61.28	11.94	35.19	156.29	31.87	11.47
orb10	105.87	42.29	60.69	19.43	44.38	166.57	16.30	12.81
orb2	102.28	35.41	64.62	22.79	36.55	154.50	8.64	11.44
orb3	112.83	36.25	69.71	16.20	44.18	174.34	9.35	8.22
orb4	123.42	45.53	71.36	17.54	49.27	176.90	10.81	11.07
orb5	79.81	38.24	46.05	12.02	33.04	141.87	9.65	8.94
orb6	89.42	42.26	60.01	10.98	34.72	153.59	16.24	6.66
orb7	41.32	16.34	24.63	12.79	17.50	73.25	8.59	8.16
orb8	88.23	43.45	49.62	16.32	36.74	158.55	16.15	11.55
orb9	124.89	35.63	71.08	22.17	47.02	168.75	10.57	12.07

Table 21: Experimental results on Hurink's [26] *vdata*, with a time budget of 60 seconds.

Instance	NRPA	MCTS	NMCTS	HBNMCTS	BINRPA	RandGreedy	GNRPA	ABGNRPA
mk01	5.87	2.25	6.91	2.64	2.28	11.98	9.57	0.71
mk02	4.41	1.78	3.64	2.52	2.09	10.90	4.46	0.96
mk03	31.34	10.09	15.92	10.96	11.99	45.92	4.65	6.01
mk04	10.60	3.59	7.12	2.43	3.76	16.60	0.95	0.94
mk05	17.34	5.42	8.87	4.23	5.79	22.93	1.02	2.32
mk06	17.20	6.48	9.20	1.63	6.13	22.33	16.84	1.14
mk07	22.27	7.40	14.19	5.84	9.60	37.19	5.84	2.89
mk08	54.96	16.65	26.19	9.65	15.27	54.40	0.85	5.26
mk09	47.75	15.66	27.00	10.79	18.97	57.87	6.18	7.31
mk10	42.47	14.64	18.92	12.56	12.90	52.33	6.27	8.56
k1	2.56	0.80	5.60	2.40	0.71	18.52	0.00	0.00
k2	4.10	1.70	3.32	1.02	1.49	32.59	0.00	0.25
k3	3.63	1.20	2.48	0.36	1.54	10.55	0.39	0.17
k4	9.09	2.36	3.12	0.48	2.55	32.13	0.49	0.50

Table 22: Experimental results on Kacem's [28] and Brandimarte's [6] instances-makespan standard deviation on 100 iterations of each algorithm with a time budget of 60 seconds for each iteration.

C.4 Bias deviation of ABGNRPA

Instance	edata	rdata	vdata	Instance	edata	rdata	vdata
abz5	7.59%	7.25%	5.98%	la08	22.54%	20.35%	19.87%
abz6	9.26%	8.10%	6.60%	la09	20.34%	21.44%	18.59%
abz7	8.11%	7.63%	5.97%	la10	20.46%	19.73%	21.05%
abz8	8.88%	9.14%	6.73%	la11	25.91%	21.61%	23.91%
abz9	8.03%	7.85%	6.36%	la12	24.15%	26.18%	24.78%
car1	30.84%	25.26%	24.17%	la13	22.36%	25.73%	24.25%
car2	31.51%	24.43%	34.14%	la14	24.37%	24.92%	24.37%
car3	26.02%	16.08%	18.13%	la15	25.78%	25.06%	22.60%
car4	24.94%	20.97%	25.83%	la16	10.29%	8.22%	6.50%
car5	17.79%	14.61%	11.96%	la17	11.94%	9.51%	6.95%
car6	12.24%	14.23%	23.00%	la18	10.56%	8.12%	6.54%
car7	13.74%	15.89%	9.86%	la19	10.62%	11.57%	6.86%
car8	9.92%	15.22%	7.44%	la20	12.54%	16.32%	6.74%
mt06	11.51%	15.00%	27.76%	la21	12.56%	12.41%	9.29%
mt10	13.60%	8.60%	6.73%	la22	11.23%	9.72%	9.02%
mt20	25.50%	26.30%	22.70%	la23	11.54%	11.43%	9.81%
orb1	13.32%	8.28%	7.44%	la24	15.47%	10.40%	8.61%
orb10	10.34%	9.37%	6.64%	la25	12.52%	11.76%	9.40%
orb2	10.34%	8.53%	7.50%	la26	11.65%	11.18%	11.13%
orb3	15.43%	10.19%	15.04%	la27	11.25%	11.84%	10.71%
orb4	14.24%	11.34%	7.93%	la28	11.47%	11.14%	10.70%
orb5	15.35%	10.85%	8.07%	la29	12.87%	12.97%	11.58%
orb6	11.29%	9.03%	6.68%	la30	14.23%	11.70%	10.94%
orb7	9.30%	11.90%	6.59%	la31	15.29%	14.21%	13.15%
orb8	16.88%	12.46%	9.16%	la32	14.80%	13.74%	13.86%
orb9	10.55%	8.67%	6.93%	la33	14.89%	15.29%	14.26%
la01	17.81%	17.71%	19.43%	la34	12.57%	13.28%	12.88%
la02	14.74%	23.62%	17.83%	la35	18.39%	14.15%	13.42%
la03	20.57%	32.14%	23.73%	la36	8.01%	7.20%	5.34%
la04	17.60%	22.65%	20.03%	la37	8.55%	7.52%	5.36%
la05	18.50%	17.85%	21.06%	la38	10.90%	9.46%	5.83%
la06	21.57%	24.75%	20.17%	la39	7.64%	7.59%	5.62%
la07	24.26%	23.51%	19.78%	la40	8.58%	7.89%	5.30%

Table 23: Bias deviation between the start and the end of an ABGNRPA run in percentages for Hurink's instances [26].

Instance	Deviation
k1	23.59%
k2	28.06%
k3	18.43%
k4	13.38%
mk01	22.79%
mk02	18.71%
mk03	13.52%
mk04	21.80%
mk05	21.05%
mk06	10.19%
mk07	30.07%
mk08	13.09%
mk09	10.87%
mk10	11.01%

Table 24: Bias deviation between the start and the end of an ABGNRPA run in percentages for Kacem [28] and Brandimarte [6] instances.