Understanding the Impact of Value Selection Heuristics in Scheduling Problems

Tim Luchterhand
LAAS-CNRS, Toulouse, France
Université de Toulouse, France

Emmanuel Hebrard
LAAS-CNRS, Toulouse, France

Sylvie Thiébaux
LAAS-CNRS, Toulouse, France
Université de Toulouse, France
Australian National University, Canberra, Australia

Abstract

It has been observed that value selection heuristics have less impact than other heuristic choices when solving hard combinatorial optimization (CO) problems. It is often thought that this is because more time is spent on unsatisfiable sub-problems where the value ordering is irrelevant. In this paper we investigate this belief in the scheduling domain and come up with a more detailed explanation. We find that, even though there are less relevant choices to be made on hard instances, each mistake tends to have a bigger impact, to a point where the potential gain from a value heuristic predominates. Moreover, we observe two interesting and relatively surprising phenomena when solving scheduling problems. First, the accuracy of a given value selection heuristic decreases with the optimality gap. Second, the computational penalty of a mistake increases with the accuracy of the heuristic. For the first observation, we argue that on hard problems, constraint propagation removes a large portion of choices that align with the intuition behind the heuristic. This means that the heuristic faces mostly difficult choices. For the second observation, we argue that simple heuristics tend to make more mistakes on intuitive choice points, and the computational cost for refuting these mistakes is smaller than for those made by a more accurate heuristic.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Computing methodologies → Discrete space search; Computing methodologies → Heuristic function construction; Computing methodologies → Neural networks

Keywords and phrases Scheduling, Branching Heuristics, Constraint Programming

Digital Object Identifier 10.4230/LIPIcs.CP.2025.27

Supplementary Material *Software (Source Code)*: `https://gitlab.laas.fr/roc/emmanuel-hebrard/tempo`, archived at `swh:1:dir:bbab6415fb8f44fef04878de35e7b50e797596bd`

Funding This work was supported by the Artificial and Natural Intelligence Toulouse Institute (ANITI) under the grant agreement ANR-23-IACL-0002, and received funding from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.

1 Introduction

Solvers for combinatorial optimization (CO) problems often strongly rely on branching heuristics that help explore an exponentially large search space efficiently. Usually, these can be categorized into two subgroups: *variable* selection and *value* selection heuristics. The former type selects the next choice point on which a branching decision should be made. The latter type then selects the actual value the variable will take. It is well known that the variable ordering can have a significant impact on the overall performance of the solver.

This is a reason why over the last decades this research area has enjoyed great popularity, and many effective variable selection heuristics like Variable State Independent Decaying Sum (VSIDS) [31], weighted degree [6] or Learning Rate Branching (LRB) [27] have been proposed. More recently, there have even been advances using machine learning (ML) and deep learning techniques [16, 22, 24].

While there exists work on value selection heuristics, the field did not receive as much attention as variable ordering [5]. The consensus arising from early works (e.g. [17, 18]) is to choose values that lead to the least constrained branches as opposed to variable selection, where usually the most constrained variable should be visited first. So independently of the problem at hand, designing a good value selection heuristic revolves around estimating the future constraining effect of a particular choice of value. The crucial part with these look-ahead value selection strategies is to find a trade-off between accuracy and computational cost. Later works leverage ideas from variable selection like VSIDS and record statistics about values during search. These include activity and impact based heuristics [30, 34] as well as the survivors-first approach of [42]. While the aforementioned heuristics perform better than a random value selection, the impact of other heuristic choices like variable ordering heuristics, nevertheless, dominates the overall performance of the solver. Still, in theory, a perfect value selection would render tree search completely obsolete [15]. One could therefore legitimately expect the potential gain from clever value selection heuristics to be significant. Additionally, in optimization problems, reaching solutions with better objective values faster strengthens constraint propagation based on the dual bound. This in turn can notably reduce the size of the overall search space. Yet, as opposed to variable selection, hardly any generic value selection heuristic has ever been adopted. Even domain-specific value selection strategies that are shown to clearly outperform random selection are relatively rare. For instance, we conducted comprehensive experiments on value selection heuristics on scheduling problems. The results in Figure 1 all show a clear trend: a clever value selection can pay off on easy problems but only marginally outperforms random choices on tight problems.

It has been argued that the shortfall of value selection heuristics often is a high computational cost in the case of look-ahead strategies [42]. But more generally, the usual explanation for the relatively low impact of value selection heuristics, especially on difficult problems, is the fact that the solver spends a lot of time in UNSAT regions of the search space. In these regions, the order of the values explored does not matter [15][1].

In this paper however, we argue that this cannot be the sole reason for the diminishing impact of value selection. While we confirm that on problems close to the phase transition [9] the solver spends most of the time in UNSAT subtrees, we also found that the remaining relevant choices tend to be crucial in finding a solution. This goes to a point where the penalty for getting these choices wrong dominates the search performance. As a consequence, a consistently accurate value selection heuristic that can avoid those errors would have a noticeable positive impact on the overall performance, especially on difficult instances.

Furthermore, we observed two interesting and rather surprising phenomena when solving scheduling problems as a sequence of satisfaction problems with an increasingly tighter bound. The first concerns the *online accuracy* of value selection heuristics which we define as the ratio between the number of *correct* branches chosen and the number of all *relevant*

[1] It may have an indirect influence via clause learning or weight-based heuristics. Nevertheless, in both cases there is no known positive correlation with some parameter that could help us decide a priori which branch is best.

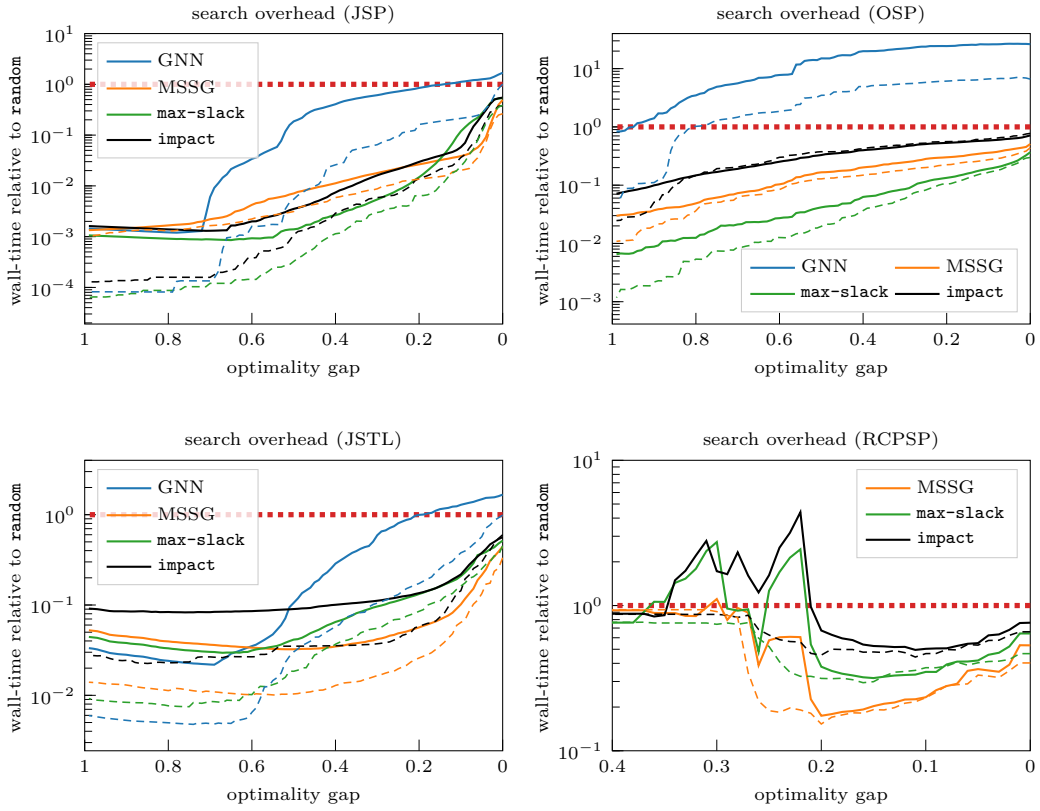

Figure 1 Average / median solve time with respect to `random` value selection on different sets of scheduling problems. Dashed lines represent median values. A point (x, y) is interpreted as: "heuristic h takes y times as much time as `random` to get to a gap of x". The red dotted line marks the performance of the `random` baseline. Details about the heuristics and the problem instances can be found in Section 3.1.

choice points. We say a choice point is relevant when there exists both a value selection for which the resulting subtree contains a solution and one for which it doesn't. A correct choice obviously occurs when the heuristic selects the value that spans a SAT subtree. In our experiments, we found that this accuracy degrades for all tested, informed heuristics as the optimality gap decreases and the problem becomes more constrained. This again means that the heuristics tend to be less accurate when the stakes are highest, i.e. when the gap is small and in turn the cost for making the wrong decision is higher than when the gap is large. Secondly, we noticed that the cost for making a wrong decision does not only depend on the difficulty of the problem but also on the accuracy of the heuristic. Our experiments suggest that, on the one hand, simple heuristics usually make more mistakes than more sophisticated ones, especially on *easy choices*, i.e. choices that follow a clear intuition. Yet, these errors are just as easily refuted, meaning that the cost for failing is low. More intelligent heuristics on the other hand, make less mistakes but when they do, it usually takes disproportionately more effort to notice and correct the error.

To support these claims, we present an in-depth analysis of the behavior of different value selection strategies on a large set of scheduling benchmarks with both disjunctive and cumulative resources. We additionally demonstrate that the mentioned phenomena also arise in experiments with an ML-based heuristic.

Our results show a more complete picture on why developing impactful value selection heuristics is difficult and identify an interplay of three key observations. First, as the optimality gap decreases, the penalty for selecting a wrong value increases. This is because wrong branches lead to increasingly deep UNSAT subtrees. Conversely, when the gap is large, these UNSAT subtrees tend to be a lot more shallow. So in theory, a heuristic with constant value prediction accuracy should have more impact the harder the problem gets because selecting the correct values avoids spending a long time in uninteresting regions. Which brings us to the second observation, namely the fact that the heuristic accuracy does *not* stay constant during the search but rather decreases with the optimality gap. This means that any heuristic will not only face more important choices but also get them wrong more often. And finally, the penalty for a wrong decision positively correlates with the accuracy of the heuristic, i.e. more involved heuristics tend to make less errors in total, but pay a disproportionately higher price for being wrong.

The remainder of this paper is structured as follows. We first introduce the metrics we used in our experiments as well as our procedure for measuring them. We then present the different heuristics we analyzed and the details of our experimental setup. Finally, we conclude the paper with a discussion of the results.

2 Analyzing the Behavior of Value Selection Heuristics

Within the scope of this paper we analyze the impact of value selection strategies on solving different types of scheduling problems. We employ a self-developed constraint programming (CP) solver we call `tempo`[2]. It represents problems using binary and numeric variables. In the case of scheduling problems, numeric variables are used to model events in a simple temporal network (STN) [11]. Each task t is composed of two events: a start event s_t and an end event e_t. Constraints are represented by edges in the form of maximum distances between the events. For example, given two events a, b, the constraint $a \prec b$ (a before b) is represented by an edge starting in b and ending in a annotated with a negative distance. Furthermore, numeric variables are also used to model resource consumption and capacities in the case of cumulative scheduling problems. The objective is modelled as one global schedule interval that must contain all tasks. The objective variable is the duration of said schedule. As many constraint optimization solvers [33, 29, 32, 10], `tempo` minimizes the objective value by solving a sequence of satisfiability problems with an increasingly tighter upper bound on the objective variable.

Binary variables on the other hand represent the existence of edges in the STN and are used as branching points during search. For example, in a disjunctive scheduling problem the binary variable $x_{t,t'}$ may represent the ordering of two tasks t and t'. Concretely, $x_{t,t'} = 1 \Leftrightarrow e_t \prec s_{t'}$, i.e. t must finish before t' starts, and $x_{t,t'} = 0 \Leftrightarrow e_{t'} \prec s_t$. `tempo` makes binary branching decisions which have been shown to be superior to d-way branching in constraint satisfaction problem (CSP) solvers [21]. This means that a binary search tree is spanned where at each node a variable is selected by the *variable* selection strategy and then assigned by the *value* selection heuristic. `tempo` uses the state-of-the-art variable selection strategy VSIDS [31]. It also leverages clause learning or no-good-learning [28] and dynamic restarts [19]. Whenever a literal appears in a conflict clause or in a resolution step leading to said clause, its activity value is updated. When clause learning is switched off, `tempo` uses

[2] The public repository can be found at `https://gitlab.laas.fr/roc/emmanuel-hebrard/tempo`. The experiments were conducted using the commit identified by the tag `cp_2025_val_selection_exp`.

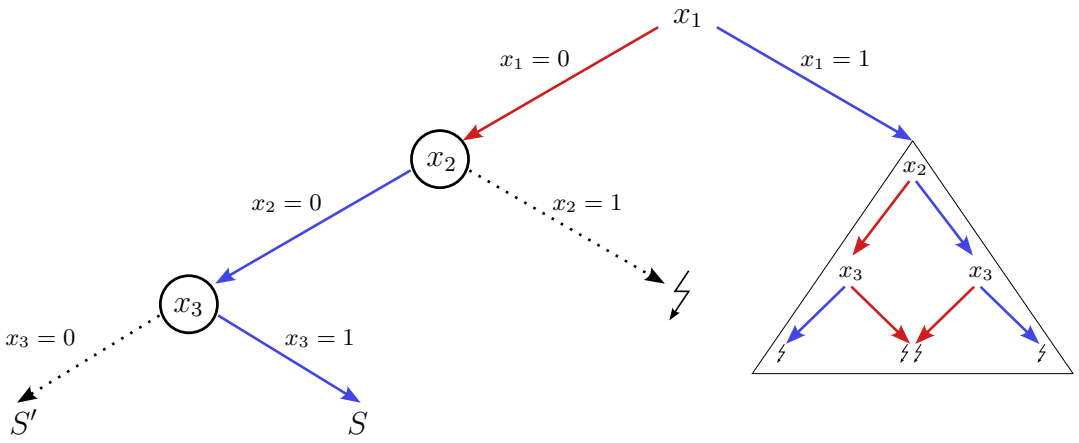

Figure 2 Depiction of the analysis procedure. Blue arrows are heuristic choices, red arrows are refutations. Circled variables need to be classified as either ICS or *relevant*. Only the choice points x_1 and x_2 are relevant. The ↯ symbols signify UNSAT branches. The triangle shows a maximal UNSAT subtree. Note that the dashed arrows are not actually explored by `tempo` but need to be visited for the classification.

explanation based weighted degree [6] instead of VSIDS for variable selection. This is possible since each fail still generates a clause that we use to update literal weights. Finally, we use an edge-finding algorithm based on [40] for propagating the disjunctive resource constraint. On cumulative resources we use an algorithm based on [26]. Without going into detail, it leverages overlaps between tasks. These are usually difficult to detect from numeric variables representing start times of tasks but much easier from ordering variables.

All in all, `tempo` achieves state-of-the-art performance on the problems treated in this paper. Using a custom solver allows us to easily access detailed information that we need for our analysis such as when and where conflicts are encountered and which specific choices were made by the heuristics.

2.1 Relevant and Irrelevant Heuristic Choices

Since `tempo` uses two-way branching, solving each satisfaction problem produces a binary search tree. When a solution is found the search is restarted from the root of the tree so that the new tighter upper bound constraint does not have to be retroactively applied to the search states. The overall search tree is then the concatenation of the trees corresponding to each satisfiability problem. Each of these search trees has the form shown in Figure 2. We measure the performance of a given value selection heuristic in each satisfiability problem by analyzing the corresponding search tree. To that end, we categorize the choice points. The branch chosen by the value selection heuristic is called a *choice* (blue arrows in Figure 2). A *refutation* is the opposite of a choice (marked with red in the illustration). A choice is *correct* when it spans a subtree that contains a solution and incorrect when it doesn't. However, some of these choices, correct or not, may be irrelevant. We say a choice is *relevant* if and only if its refutation would have lead to a different outcome. That is, if the subtree spanned by the choice contains a solution and the one spanned by the refutation does not or vice versa. Consequently, all choices in an UNSAT subtree are irrelevant because neither of the branches leads to a solution. This type of choice is called irrelevant choice in an UNSAT subtree (ICU). Moreover, some correct choices are irrelevant if the refutation also leads to a

solution. These are called irrelevant choices in a SAT subtree (ICS's). In theory, one could further distinguish correct choices, i.e. whether both branches lead to equally good solutions (in terms of objective value) or not. However, we are primarily interested in the solver not getting stuck in UNSAT subtrees and therefore do not make this distinction.

2.2 Value Selection Heuristic Accuracy

Each time `tempo` finds a solution S we get the following information.

1. The total number of heuristic choices T in the current descent. In Figure 2 this is simply the number of blue arrows.
2. The number of correct choices C, i.e. two in the example.
3. The length L of the branch leading to the solution S (three in the example). Note that L is in general less than the number of variables since constraint propagation will fix some variables.
4. The number of ICUs (the three blue arrows in the triangle in Figure 2) which is simply $T - L$.

In order to obtain the number of ICS's we need to further analyze the correct choices (circled variables in Figure 2). To analyze a choice on a variable x, we define a satisfiability problem based on the initial problem description while adding all choices and refutations leading from the root to x as constraints. For example, to classify x_3 in Figure 2 we would add the constraints $x_1 = 0$ and $x_2 = 0$ as well as the previous upper bound on the objective and then call a satisfiability oracle. If the oracle determines that the problem is SAT, i.e. by finding a different improving solution S' (as in the figure), the choice is an ICS. Having obtained the number of ICS's, we can finally calculate the number of *relevant* choices

$$R = T - \#\text{ICU} - \#\text{ICS} = L - \#\text{ICS} \tag{1}$$

and the number of relevant and correct choices

$$R_C = C - \#\text{ICS}. \tag{2}$$

In our running example, we would get $R = 2$ and $R_C = 1$. Based on this information, we then define the *accuracy* of a value selection heuristic h simply as the ratio of relevant and correct choices to the number of relevant choices:

$$\text{accuracy}(h) = \frac{R_C}{R}. \tag{3}$$

In the example shown in Figure 2, the accuracy would be 50%, given that there are two relevant choices of which one is wrong. We can monitor this accuracy as the search progresses since `tempo` logs the required information each time an improving solution is found. We call the progression of the accuracy over the optimality gap *online* accuracy. Furthermore, we are interested in the cost of a wrong relevant choice. We define this *error penalty* P proportional to the average size of a maximal UNSAT subtree which is simply the number of ICUs divided by the number of mistakes. Since we want to compare this number across problems of different size, we normalize it by additionally dividing by the total number of choices. Thus, we get

$$P = \begin{cases} \frac{\#\text{ICU}}{T(R-R_C)}, & \text{if } R_C < R, \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

In Figure 2, the average size of maximal UNSAT subtrees would be three and $P = 0.5$. This measure gives us an indication of how bad an error by the heuristic is on average, i.e. how much time is wasted in UNSAT regions.

While our procedure gives us detailed information about the behavior of the value selection heuristic, it is also computationally expensive since each oracle call involves solving a SAT problem. Still, the number of oracle calls is only $\mathcal{O}(L)$ and L is typically much smaller than the number of variables due to constraint propagation. Also notice that this procedure is not exact on search trees developed by conflict driven clause learning solvers. This is due to the fact that the refutation of a choice $x = v$, where v is some value, is not necessarily $x \neq v$. More precisely, a backjump from the choice $x = v$ may jump to a choice point on a different variable y. This means that the decision $x = v$ was not necessarily wrong. Also, all the decisions strictly between the backjump level and the decision level are not well categorized. Therefore, when applying this method to a search tree using learning, one should be aware that the results are only approximations.

3 Experiments

In our experiments we analyzed the behavior of different value selection heuristics on a set of scheduling problems. These can be viewed as a sequence of satisfiability problems that get progressively harder as the upper bound on the objective becomes tighter. In accordance with the theory on phase transitions [9] we identify the optimality gap of our objective value as an order parameter and argue that for small values the resulting satisfiability problems are close to the phase transition. This is reasonable given that the number of solutions and thus the probability of finding one decreases with the gap.

The remaining section is structured into five parts. First we present the studied heuristics and our experimental setup. In Sections 3.2 and 3.3 we then continue with a detailed analysis on the progression of the number of irrelevant and relevant choices during search. Next, we move on to our main observations in Section 3.4. And finally, we conclude our experiments by discussing our observations and their implications.

3.1 Setup

In our experiments we analyzed four different heuristics. In the following, we explain their behavior given a binary variable x whose values 1 and 0 correspond to the event precedence constraints $a \prec b$ and $c \prec d$ respectively. As a baseline we use the `random` heuristic that, as the name suggest, chooses values randomly with uniform probability.

First, the `max-slack` heuristic uses the following branching rule:

$$\texttt{max-slack}(x) = \begin{cases} 1, & \text{if } \max(b) - \min(a) \geq \max(d) - \min(c) \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

where $\min(a)$ and $\max(a)$ are the minimum and maximum values of the domain of the numeric variable a respectively. Intuitively, the heuristic fixes the edge that leaves the largest possible slack in the temporal network. The variable $x_{t,t'}$ in a disjunctive scheduling problem for example represents the ordering of the tasks t, t': $x_{t,t'} = 1 \Leftrightarrow e_t \prec s_{t'}, x_{t,t'} = 0 \Leftrightarrow e_{t'} \prec s_t$. `max-slack` would therefore schedule $t \prec t'$ if $\max(s_{t'}) - \min(e_t) \geq \max(s_t) - \min(e_{t'})$.

Second, we use an impact based heuristic similar to [34]. It keeps a score $i_{\mathtt{x}}$ for each binary literal $\mathtt{x} \Leftrightarrow x = 1, \neg\mathtt{x} \Leftrightarrow x = 0$ in the problem, which is initially zero. Whenever the heuristic assigns a value to x, we count the number of other variables fixed by subsequent constraint propagation and update the score[3]:

$$i_{\mathtt{x}}^{(k+1)} = \frac{i_{\mathtt{x}}^{(k)} + \#\text{ variables fixed by propagation}}{2}. \tag{6}$$

This score is used by the `impact` heuristic to make decisions according to

$$\texttt{impact}(x) = \begin{cases} 1, \text{ if } i_{\mathtt{x}} < i_{\neg\mathtt{x}} \\ 0, \text{ otherwise.} \end{cases} \tag{7}$$

The intuition is somewhat similar to that behind `max-slack`: fix variables in a way that leaves the most freedom for future decisions. The problem with using this rule right from the beginning is that the impact scores are all zero. Our actual implementation of `impact` therefore only follows Equation (7) after a short warm-up period, i.e. after 50% of all literals have been selected at least once, and falls back on `max-slack` otherwise.

Next, we combine `max-slack` and solution-guided search [4] which we call `max-slack` solution guided (MSSG). Initially, MSSG uses `max-slack` until a solution $S = (x_i)_{i=1}^{N}$ ($N \in \mathbb{N}$ the number of binary variables) is found. This solution is then saved. From this point on, MSSG tries to keep the variable assignment of S. Since at some point the search needs to deviate from the path to the last solution due to the tighter upper bound, MSSG is permitted to fall back to `max-slack` if the discrepancy between the S and the current variable assignment $\tilde{S} = (\tilde{x}_i)_{i\in I}, I \subset \{1, \ldots, N\}$ becomes too big. This discrepancy is defined as

$$\text{discrepancy}\left(S, \tilde{S}\right) = \begin{cases} 0, \ I = \emptyset, \\ \frac{1}{|I|}\sum_{i\in I} 1 - \delta_{x_i\tilde{x}_i}, \ I \neq \emptyset \end{cases}, \ \delta \text{ the Kronecker delta,} \tag{8}$$

i.e. the percentage of assigned variables whose values differ from the corresponding ones in S. In our experiments we set this discrepancy threshold to 1%. So in short, MSSG behaves the following way:

$$\text{MSSG}(x) = \begin{cases} \texttt{max-slack}(x), \text{ if no solution found yet or discrepancy } > 1\%, \\ 1, \text{ if } \mathtt{x} \in S, \\ 0, \text{ if } \neg\mathtt{x} \in S. \end{cases} \tag{9}$$

Each time a new improving solution is found, S is updated.

Finally, inspired by the works of [38, 39], we tested a heuristic based on a pre-trained graph neural network (GNN). Without going into too much detail, the GNN exploits a graph representation of the problem, much like the STN. Nodes represent tasks as well as resources, edges between tasks describe precedence relations and are annotated with timing information. Edges between tasks and resources describe resource dependencies and are also annotated with features like resource demand and capacity. From the graph, the GNN calculates a graph embedding using a message passing scheme similar to that in [3]. From this embedding the binary values for each edge are derived using a multi-layer perceptron (MLP). Since running the GNN is costly, the inference is only run at the very beginning of the search. Additionally, it is rerun whenever a solution is found which tightens the upper bound on the objective.

[3] Our implementation differs from that of the original paper for reasons of simplicity.

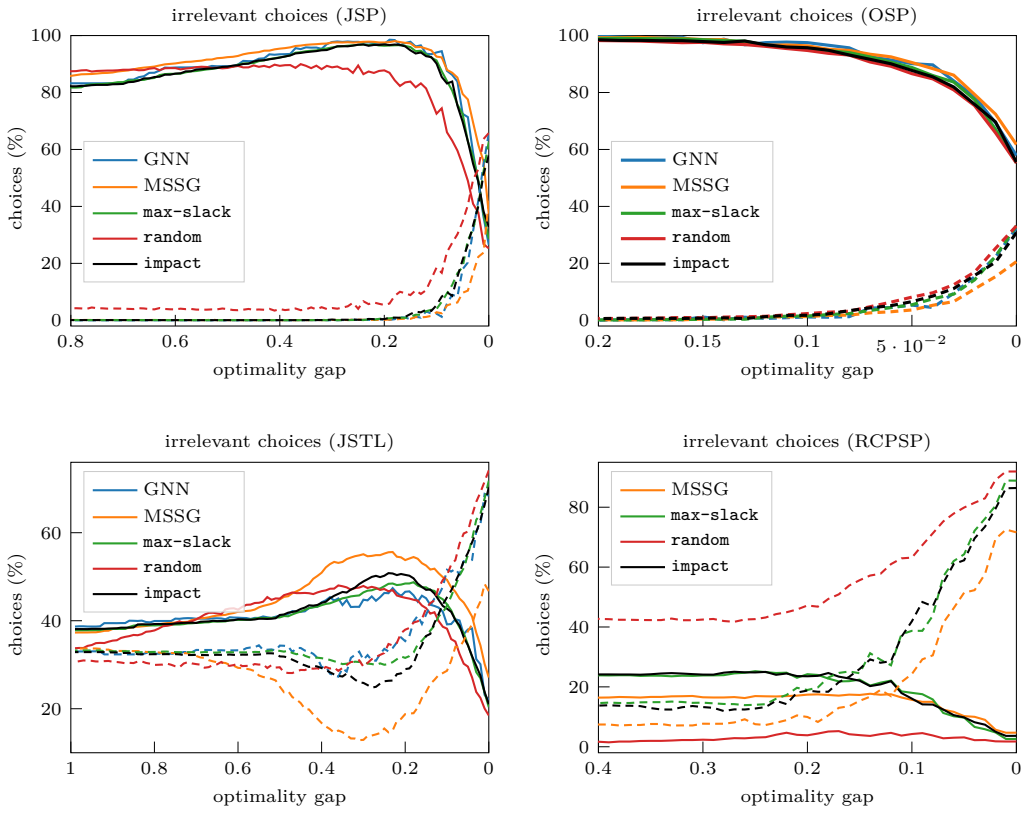

Figure 3 Mean percentage of irrelevant choices distinguished by category. Bold lines represent the percentage of ICS's, dashed lines represent the percentage of ICUs.

We evaluated these heuristics on four sets of different scheduling problem types: 48 Open Shop Problems (OSPs) from [7, 20, 43], 24 Job Shop Problem (JSP) instances from [1, 14, 2, 25, 37, 43, 41], 58 Job Shop instances with time lags (JSTL) from [8] and 20 resource-constrained project scheduling problem (RCPSP) instances from [23]. A complete list of the instances used is given in Table 1. We ran `tempo` with each of the five heuristics on each of those instances with ten different random seeds, totaling in 1500 runs for each heuristic. An exception to this is the GNN based heuristic that, at the time of the experiments, could not handle cumulative resources, which is why we do not present results for the GNN on RCPSPs. Each run was given a timeout of 30 minutes. The results in the next section were produced without using clause learning in order to avoid the problem mentioned Section 2.2. However, results with clause learning can be found in Section A.

3.2 Progression of Irrelevant Choices

We begin our analysis by addressing the belief that the solver spends increasingly more time in UNSAT subtrees as the optimality gap closes. Even though our results seem to align with this hypothesis, they draw a more complete picture as we analyze ICUs and ICS's separately. Figure 3 shows the progression of the share of irrelevant decisions over the course of solving the problem.

First, we can see that, in all four problem types, the percentage of ICUs (dashed lines) increases towards smaller gaps. At the same time, the percentage of ICS's (bold lines) decreases. It is also visible that this effect depends on the type of problem. On less constrained instances like OSPs or JSPs, initially nearly all choices lead to improving solutions and are therefore irrelevant. This is most pronounced on OSPs due to their symmetry. When the gap decreases, the upper bound on the objective variable tightens, making the problem more constrained and ultimately leading to increasingly more failures and therefore more ICUs. Interestingly, on JSP instances, the ratio of ICS's actually appears to increase slowly before falling back off towards the end (not counting `random`). One possible explanation is that propagation removes more and more choices and helps the heuristics guide the solver into *easy regions* where most choices are ICS's, as long as the gap is not too small.

The same overall trend can be observed on the JSTL and RCPSP instances. However, the levels of ICS's and ICUs are initially much closer together. In JSTLs there exist backwards links between tasks in the STN due to the time lags. These can easily cause negative timing cycles and thus failures which explains the higher initial percentage of ICUs and the lower amount of ICS's. For the RCPSP instances, both types of irrelevant choices initially make up a comparatively low share or conversely: there are more relevant choices than in any other problem type in the beginning of the search. This is due to a particularity of the resource model used that only performs very limited forward checking. To give an example, let's say we have a resource with capacity 10 and three tasks t_1, t_2, t_3 that consume quantities of 3, 5 and 6 resource units respectively. Let's also say that t_1 and t_2 are currently assigned to run in parallel. The model does not forbid to also schedule t_3 in parallel, even though it would result in a trivial failure. This means that a lot of relevant but rather trivial choices are created.

Perhaps more interestingly, the amount of irrelevant choices not only depends on the gap but also on the heuristic itself. Looking at `random` for example, it produces the most ICUs and the least ICS's on both JSP and RCPSP instances. This makes sense because `random` is arguably a bad heuristic that makes a lot of mistakes. As a result, the solver will spend comparatively more time in UNSAT regions, leading to more ICUs. Conversely, `random` is less likely to steer the solver into regions where any choice leads to a solution. This effect, while less pronounced, is also present in the JSTL instances, at least towards smaller gaps. This is probably due to the fact that all heuristics have a harder time making good decisions on this type of instance which is why the difference is less noticeable. The same can be seen on the OSP instances, albeit for the opposite reason. As mentioned before, OSPs have a high degree of symmetry, meaning that when the gap is large basically any decision, random or not, leads to a solution. MSSG on the other hand behaves quite the opposite way. As we will see later, it has one of the highest online accuracies. With this in mind, it seems logical that it exhibits high ICS and low ICU percentages on all four problem types.

3.3 Progression of Relevant Choices

Our observations thus far coincide with what is accepted in the literature. Things become more interesting when we combine both the curves for ICS's and ICUs, or put differently, when we look at the number of *relevant* choices. This can be seen in Figure 4. The results on the RCPSP are arguably the most intuitive. The amount of relevant choices starts out rather high and then progressively diminishes as the gap closes. The opposite can be seen on the OSP instances. Again, it is apparent that on these problems, initially virtually everything is a solution and choices only start to matter when the gap becomes small. On both types of Job Shop problems, the results are a bit more complicated to interpret, and the differences

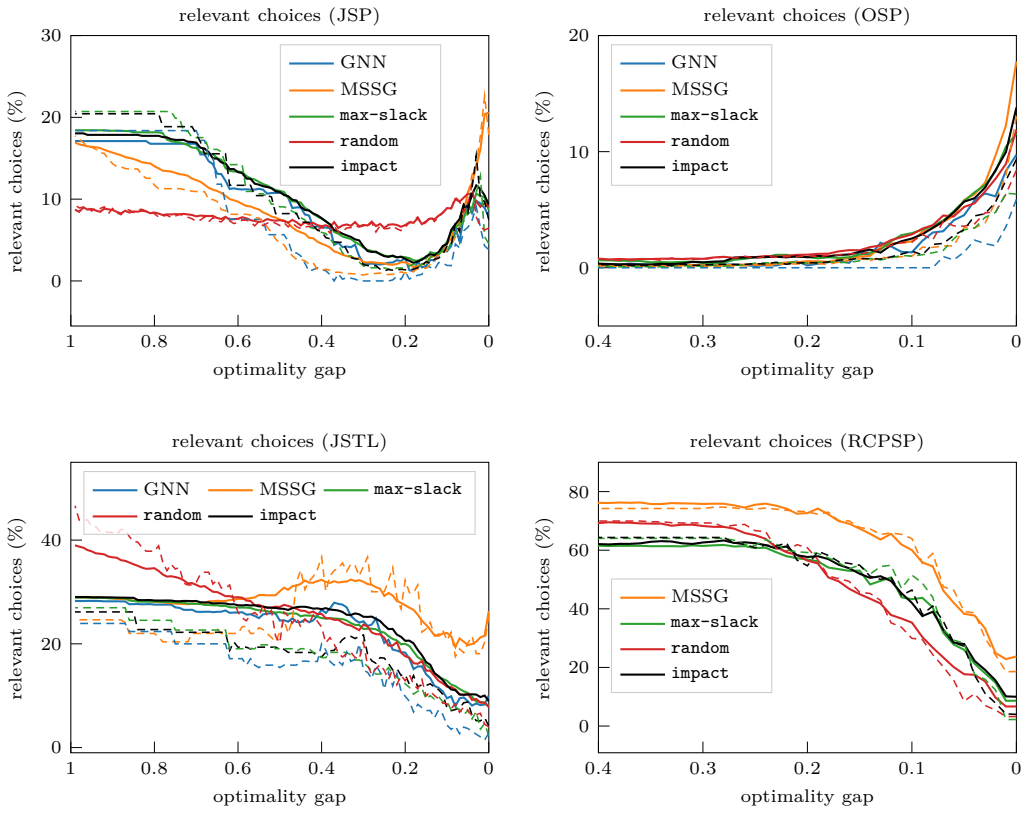

Figure 4 Percentage of relevant choices. Bold lines represent mean values and dashed lines median values.

between the heuristics are more pronounced. On the JSTLs the percentage of relevant choices still follows a clear downward trend except when using solution guided search. Interestingly, `random` has the highest rate of relevant choices in the beginning. Comparing its graph with the one form Figure 3, we can see that this is due to `random` exhibiting both the lowest percentage of ICS's and ICUs. It appears logical that `random`, being a bad heuristic, rarely guides the solver into a subtree with many ICS's. Nevertheless, it seems strange that it initially also exhibits the lowest level of ICUs. One possible explanation could be that `random` makes a lot of trivial mistakes that are easily refuted. Consequently, the solver doesn't spend a long time in an UNSAT region after a wrong decision. MSSG on the other hand, maintains a relatively constant level of relevant choices over the whole range of gap values. There is even a part where the percentage of relevant choices increases over the initial level. Again, comparing the corresponding curves in Figure 3 we can see that this arises from a very low level of ICUs and implies that MSSG provides good guidance. When the gap approaches zero, the number of relevant choices remains at about the same level as in the beginning. At this point, there remain much fewer ICS's in the overall search space. The fact that the number of relevant choices remains high means that MSSG manages to keep the search in regions where it is still possible to find improving solutions, albeit only through clever choices. Finally, on the JSP instances there is a clear difference between `random` and the other heuristics. As on the OSPs, these problems are not very constrained when the gap is small which means that most of the choices are ICS's. The difference between `random` and

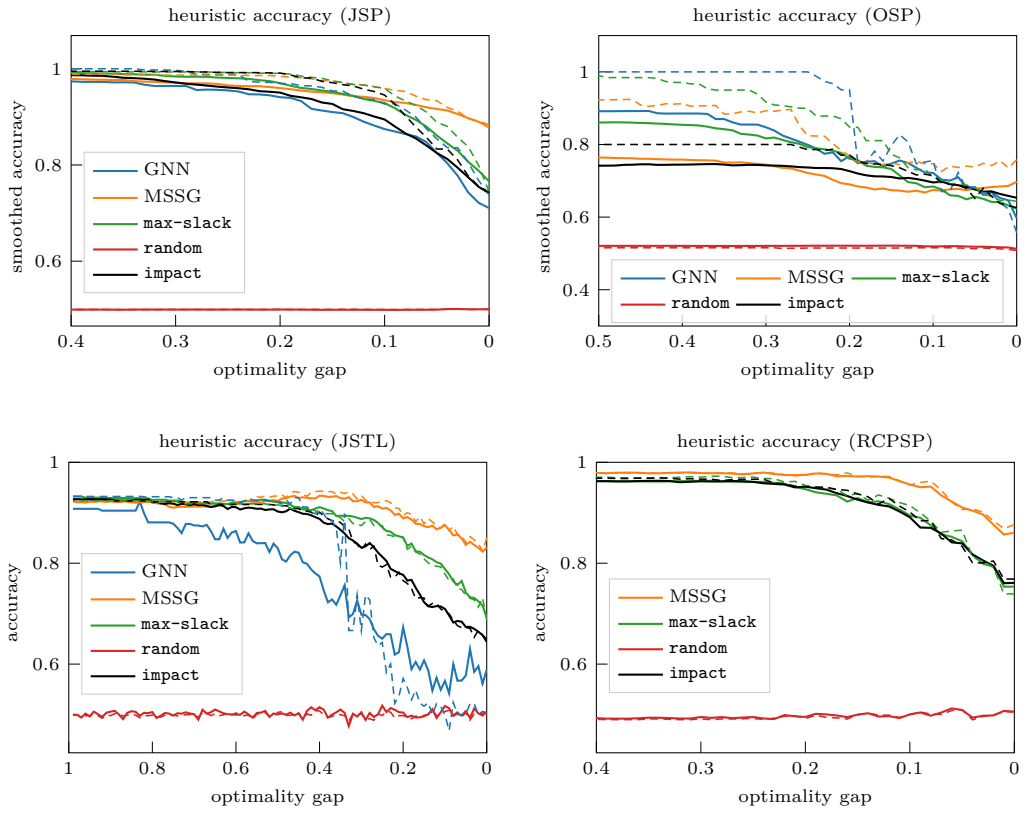

Figure 5 Online accuracy over optimality gap. Bold lines represent mean values and dashed lines median values. In the two top figures, the accuracy has been smoothed using a biased average.

the other heuristics arises from the fact that `random` makes more errors and thus also has a higher level of ICUs right from the beginning. What is worth of notice is that for very small gap values, the percentage of relevant choices actually rises back up, in the case of MSSG even higher than the initial ratio.

To summarize the results this far, we confirm that all heuristics spend more time in UNSAT subtrees the smaller the gap becomes. Still, the distinction between ICS's and ICUs draws a more complete picture and reveals more details about the behavior of the different heuristics across the four instance types. The results also differ somewhat from the intuition in the literature, especially when looking at the number of relevant choices. Only on RCPSP instances we observe a clear downward trend of the heuristic's impact as the gap shrinks. On the other problem types, the trend is either less clear (JSTL), inverses towards the end (JSP) or goes in the complete opposite direction (OSP). On the latter three types, one would expect the heuristics to have a greater impact on the performance of the solver, provided of course that their accuracy does not change. However, the results we will discuss next show that this is precisely not the case.

3.4 Online Accuracy and Error Penalty

In the following part, we only focus on the relevant choices and analyze the actual online accuracy of the heuristics presented in Figure 5. As expected, `random` has a constant accuracy of 50% all the time on all four problem types. All the other heuristics on the other hand

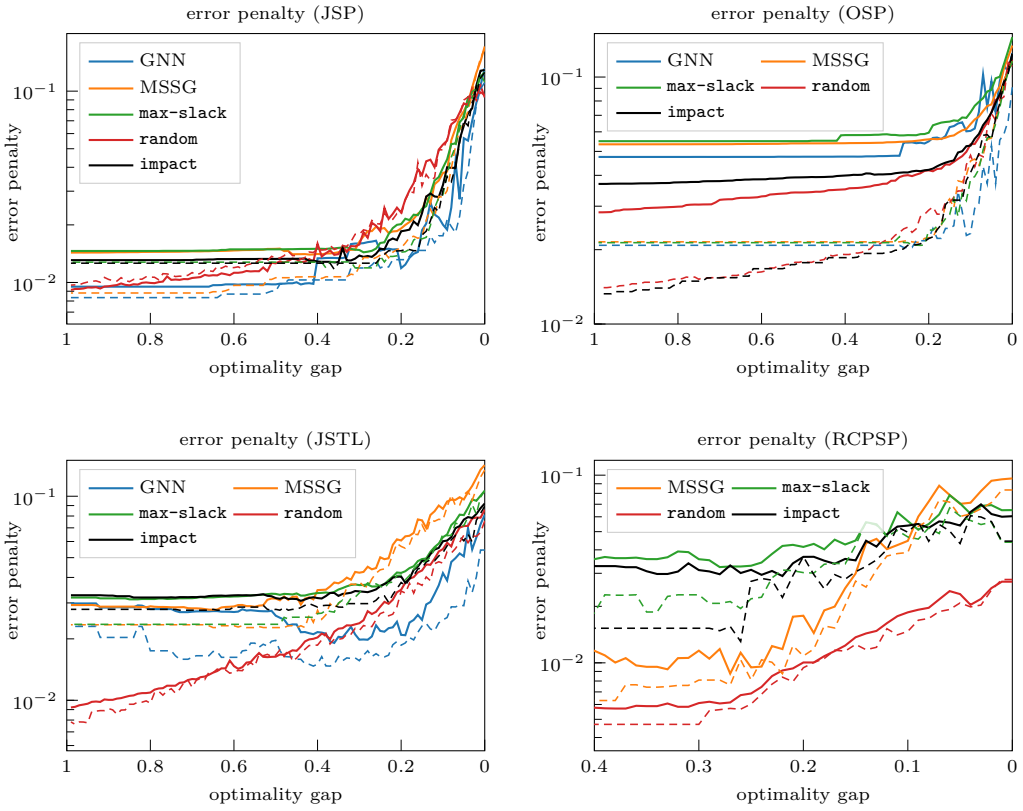

Figure 6 Error penalty over optimality gap. Bold lines represent mean values and dashed lines median values.

show a clear drop in accuracy as the gap becomes smaller. MSSG seems to suffer the least from this problem. So apparently, the relevant choices the heuristics face become more and more difficult over the course of the search. We hypothesize that this happens due to an *alignment* between constraint propagation and value selection heuristics. In short, constraint propagation often prunes decisions that are in line with the heuristic. Thus, when the problem is tight and constraint propagation is strong, fewer intuitive decisions are left to the heuristic which ultimately results in a lower accuracy. We go into more detail in Section 3.5.

Finally, not only does the heuristic accuracy decrease with the gap, it is also lowest when the stakes are highest. Figure 6 shows the progression of the error penalty for the different heuristics. As expected, this penalty increases towards smaller gap values on all instances and for all heuristics. This means that the average size of the UNSAT subtrees becomes bigger and mistakes generally occur earlier in the search. In some cases, we can observe an increase of about a factor of ten (MSSG on JSP or RCPSP). This observation is in line with the theory on phase transitions [9]. There exists work on the notion of exceptionally hard problems [35], i.e. problems that are located to the far left of the phase transition and still are very difficult. However, we did not encounter them in our experiments. A possible explanation for this is that their difficulty is not necessarily a property of the problems but also depends on the algorithm and heuristics used [36].

So even though on some instance types the percentage of relevant choices decreases as we approach the optimum, the remaining relevant choices tend to be far more impactful. Moreover, the error penalty also depends on the heuristics themselves. When comparing Figure 5 and Figure 6, it becomes clear that the most accurate heuristic (MSSG except on OSP instances) also suffers the biggest error penalty. In contrast, while the error penalty for `random` also increases, it usually starts out lowest of all heuristics and ends at smaller values than MSSG.

For the sake of completeness, we want to mention that we also did some experiments with `tempo` with enabled no-good-learning. The results, however, are not accurate as discussed in Section 2.2. We will therefore not analyze them in detail here, but they can be found in Section A. The observations mostly coincide with those made in this chapter.

3.5 Discussion

In summary, our experiments reveal three key observations. First, while the percentage of irrelevant choices in an UNSAT subtree (ICUs) does increase towards smaller gap values, the percentage of relevant choices does not necessarily decrease at the same time for all problem types. We found it to be the case on some instance types like RCPSPs and quite the opposite on OSPs. Moreover, this effect depends on the heuristics themselves: better heuristics tend to face more relevant choices. Second, the accuracy of all informed heuristics, that excludes `random`, becomes less accurate the tighter the problem becomes. And finally, the error penalty, i.e. the time spent in an UNSAT subtree after a wrong heuristic decision, increases as the gap decreases. This effect is most pronounced for accurate heuristics.

These three phenomena and especially their interplay give us a better understanding of why designing impactful generic value selection heuristics is difficult. From our results it seems clear that the fact that the solver spends most of the time in UNSAT subtrees cannot be the only explanation for the limited impact of these heuristics on hard instances. A consistently accurate heuristic should be especially impactful since it would be able to avoid the increasingly large error penalties. This would create somewhat of a positive feedback loop. Fewer errors lead to more relevant choices that the heuristic gets mostly right given its high accuracy which in turn leads to more relevant choices and so on. This is obviously not the case. There are even instance types where the overall percentage of relevant choices does not decrease, yet still the search slows down towards smaller gap values.

We therefore present an additional argument, namely that the branching choices become increasingly non-intuitive the harder the problem gets. This degradation of the heuristic accuracy can be clearly seen in Figure 5. Our explanation for this is that heuristic branching choices often *align* with constraint propagation. For instance, the `max-slack` heuristic suggests to sequence two events in a way that leaves the most slack in the timing network, i.e. it prefers to keep large positive timing values. Notice that constraint propagation of binary disjunctions precisely excludes any ordering decision that would lead to negative slack. In general, the principle behind value branching is to first explore branches that are less likely to be inconsistent and hence less likely to be pruned. Therefore, decisions pruned by constraint propagation are overwhelmingly those that the value selection heuristic would have gotten right anyway. As a result, when the gap decreases and the tightness of the problem increases, constraint propagation gets stronger and is increasingly prevalent in the shape of the search tree which ultimately means that the heuristic will more likely make the wrong decision on the choices points that are left. Or put differently, due to the alignment with constraint propagation, decisions that go against the intuition behind the heuristic become more likely. Conversely, if the problem is not tight (large gap), constraint propagation is less strong and

more easy choices that follow a clear intuition are left to the heuristic. In future work, it would be interesting to see whether this phenomenon can be reproduced in settings where much weaker propagation is used. In our case, it seems to arise for all heuristics, be they relatively simple like `max-slack` or more complex like the GNN. A very accurate heuristic should still get more of the difficult choices right. But given the nature of the problem, a heuristic that accurately models an exponential number of choices is either impossible to design or too costly to run.

The third observation, i.e. the increasing error penalty, only amplifies the effect of the degrading accuracy. What's most noteworthy is the fact that the penalty again depends on the heuristic. As described previously, when the optimality gap is small, the more accurate heuristics pay a disproportionately high price for selecting the wrong value. Our hypothesis is that bad heuristics like `random` make more errors on the one hand that are easily refuted on the other hand. In contrast, more accurate heuristics make decisions that initially seem correct but turn out to be wrong much later in the search. In a way this effect offsets the advantage of a high accuracy and could explain why we have yet to see a superior generic value selection heuristic. It also gives a new perspective on reasoning about the effectiveness of these heuristics. The fact that one heuristic is more accurate than another on paper does not necessarily translate to a better performance in practice.

4 Conclusion and Future Work

In this paper we addressed a common belief in the literature that explains why value selection heuristics generally have a lower impact on the performance of a CO solver than other heuristic choices like for example variable selection. To that end we analyzed different value selection heuristics on a set of scheduling benchmarks. In our experiments, we were able to confirm that, when the optimality gap shrinks, the solver *does* spend more time in unsatisfiable subtrees where the value ordering effectively has no impact. We also argued that this cannot be the sole reason for the diminishing influence of value selection heuristics and presented further interesting phenomena in our results. First, the heuristics become less accurate the smaller the gap, most likely due to the interaction with constraint propagation that prunes away many easy branching choices and leaves the heuristics with mostly unintuitive decisions. What makes this worse is our second observation, namely that the penalty for selecting the wrong value strongly increases for small optimality gaps. This means that the heuristics fail most often when the stakes are highest. And finally, this error penalty also positively correlates with the heuristic's accuracy, meaning that better heuristics pay a higher price for making mistakes. These observations, on the one hand, explain why the impact of value selection heuristics diminishes on harder problems. On the other hand, they explain why the gain in performance from an accurate heuristic is not necessarily as high as expected in practice. This should be kept in mind when employing methods like ML to improve heuristic parts of the solver. While ML has been successfully used for variable selection, using it for value selection seems far less promising. In fact, we found that our GNN based heuristic was unable to provide any significant advantage – or depending on the instance type any advantage at all – over the other much simpler heuristics.

At this point we want to stress that our results neither suggest that further research on value selection heuristics is pointless nor do they contradict works on sophisticated heuristics like [13, 12]. We simply made some interesting observations that hopefully will help to better understand the impact of value selection in CO problems like scheduling. To give

some concrete advice based on our findings: if one is dealing with large industrial problem instances where finding an optimum is futile, a clever value selection heuristic that aims to quickly find a high quality solution may prove very effective. Conversely, if optimality is of interest, "dumb but fast" is probably the way to go.

Finally, we acknowledge that our results are restricted to the scheduling domain and in some parts to the solver implementation we used. Specifically, our solver `tempo` uses 2-way branching and strong propagation mechanisms, and our analysis procedure only obtains approximate results when using no-good-learning and backjumping. It should be possible to develop an algorithm that can obtain exact results even with backjumps in order to confirm our findings. Such a procedure surely exists, even though it might be more computationally demanding. More importantly however, we think it would be interesting to extend our experiments to a broader setting, i.e. general CSPs with arbitrary value domains and different phase transitions. Our explanations for the observations made in this paper are not directly tied to scheduling which is why we believe that similar phenomena might be observed on different problem types as well.

References

1 Joseph Adams, Egon Balas, and Daniel Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3):391–401, 1988. URL: `http://www.jstor.org/stable/2632051`.

2 David Applegate and William Cook. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3(2):149–156, May 1991. `doi:10.1287/ijoc.3.2.149`.

3 Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, June 2018. `arXiv:1806.01261`.

4 J. Christopher Beck. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, May 2007. `doi:10.1613/jair.2169`.

5 Christian Bessière and Jean-Charles Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Principles and Practice of Constraint Programming — CP96*, pages 61–75, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. `doi:10.1007/3-540-61551-2_66`.

6 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pages 146–150, NLD, 2004. IOS Press.

7 Peter Brucker, Johann Hurink, Bernd Jurisch, and Birgit Wöstmann. A Branch & Bound Algorithm for the Open-shop Problem. *Discrete Applied Mathematics*, 76(1):43–59, 1997. Second International Colloquium on Graphs and Optimization. `doi:10.1016/S0166-218X(96)00116-3`.

8 Anthony Caumond, Philippe Lacomme, and Nikolay Tchernev. A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research*, 35(7):2331–2356, July 2008. `doi:10.1016/j.cor.2006.11.007`.

9 Peter C. Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 331–340. Morgan Kaufmann, 1991. URL: `http://ijcai.org/Proceedings/91-1/Papers/052.pdf`.

10 Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, a lazy clause generation solver. `https://github.com/chuffed/chuffed`, 2018.

11 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. `doi:10.1016/0004-3702(91)90006-6`.

12 Augustin Delecluse and Pierre Schaus. Black-Box Value Heuristics for Solving Optimization Problems with Constraint Programming. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:12, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CP.2024.36`.

13 Jean-Guillaume Fages and Charles Prud'Homme. Making the First Solution Good! In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1073–1077, 2017. `doi:10.1109/ICTAI.2017.00164`.

14 H. Fisher and G.L. Thompson. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In *Industrial Scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, 1963.

15 Daniel Frost and Rina Dechter. Look-Ahead Value Ordering for Constraint Satisfaction Problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 572–578. Morgan Kaufmann, 1995. URL: `http://ijcai.org/Proceedings/95-1/Papers/075.pdf`.

16 Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, pages 15554–15566, 2019. URL: `https://proceedings.neurips.cc/paper/2019/hash/d14c2267d848abeb81fd590f371d39bd-Abstract.html`.

17 Pieter Andreas Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, pages 31–35, USA, 1992. John Wiley & Sons, Inc.

18 Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search Lessons Learned from Crossword Puzzles. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 – August 3, 1990, 2 Volumes*, pages 210–215. AAAI Press / The MIT Press, 1990. URL: `http://www.aaai.org/Library/AAAI/1990/aaai90-032.php`.

19 Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the 15th National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437, USA, 1998. American Association for Artificial Intelligence. URL: `http://www.aaai.org/Library/AAAI/1998/aaai98-061.php`.

20 Christelle Guéret and Christian Prins. A new lower bound for the open-shop problem. *Annals of Operations Research*, 92:165–183, 1999. `doi:10.1023/A\%3A1018930613891`.

21 Joey Hwang and David G. Mitchell. 2-Way vs. d-Way Branching for CSP. In *Principles and Practice of Constraint Programming – CP 2005*, pages 343–357, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. `doi:10.1007/11564751_27`.

22 Elias B. Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to Branch in Mixed Integer Programming. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 724–731. AAAI Press, 2016. `doi:10.1609/AAAI.V30I1.10080`.

23 Rainer Kolisch and Arno Sprecher. PSPLIB – A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, January 1997. `doi:10.1016/s0377-2217(96)00170-1`.

24 Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver? In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020. URL: `https://proceedings.neurips.cc/paper/2020/hash/6d70cb65d15211726dcce4c0e971e21c-Abstract.html`.

25 S. Lawrence. *An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. PhD thesis, Carnegie-Mellon University, 1984.

26 Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A Scalable Sweep Algorithm for the cumulative Constraint. In *Principles and Practice of Constraint Programming – 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 2012. doi:10.1007/978-3-642-33558-7_33.

27 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning Rate Based Branching Heuristic for SAT Solvers. In *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 123–140, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-40970-2_9.

28 J.P. Marques Silva and K.A. Sakallah. GRASP-A new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, ICCAD-96, pages 220–227. IEEE Comput. Soc. Press, 1996. doi:10.1109/iccad.1996.569607.

29 L. Michel, P. Schaus, and P. Van Hentenryck. Minicp: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021. doi:10.1007/s12532-020-00190-7.

30 Laurent Michel and Pascal Van Hentenryck. Activity-Based Search for Black-Box Constraint Programming Solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 228–243, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-29828-8_15.

31 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/378239.379017.

32 Laurent Perron and Frédéric Didier. CP-SAT. URL: https://developers.google.com/optimization/cp/cp_solver/.

33 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.

34 Philippe Refalo. *Impact-Based Search Strategies for Constraint Programming*, pages 557–571. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30201-8_41.

35 Barbara M. Smith and Stuart A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence – Volume 1*, IJCAI'95, pages 646–651, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

36 Barbara M. Smith and Stuart A. Grant. Where the Exceptionally Hard Problems Are. Technical report, University of Leeds, 1995.

37 Robert H. Storer, S. David Wu, and Renzo Vaccari. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management Science*, 38(10):1495–1509, 1992. URL: http://www.jstor.org/stable/2632676.

38 Florent Teichteil-Königsbuch, Guillaume Povéda, Guillermo González de Garibay Barba, Tim Luchterhand, and Sylvie Thiébaux. Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks. In *Proc. 33rd International Conference on Automated Planning and Scheduling*, pages 623–633. AAAI Press, 2023. doi:10.1609/ICAPS.V33I1.27244.

39 Hélène Verhaeghe, Quentin Cappart, Gilles Pesant, and Claude-Guy Quimper. Learning Precedences for Scheduling Problems with Graph Neural Networks. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2024.30.

40 Petr Vilím, Roman Barták, and Ondrej Cepek. Extension of $O(n \log n)$ Filtering Algorithms for the Unary Resource Constraint to Optional Activities. *Constraints An Int. J.*, 10(4):403–425, 2005. doi:10.1007/S10601-005-2814-0.

41 Takeshi Yamada and Ryohei Nakano. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. In *Parallel Problem Solving from Nature*, 1992. URL: https://api.semanticscholar.org/CorpusID:37171770.

42 Zhijun Zhang and Susan L. Epstein. Learned value-ordering heuristics for constraint satisfaction. In *Proceedings of STAIR-08 Workshop at AAAI-2008*, 2008. URL: https://api.semanticscholar.org/CorpusID:16569060.

43 Éric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. Project Management and Scheduling. doi:10.1016/0377-2217(93)90182-M.

A Appendix

A.1 Additional Figures

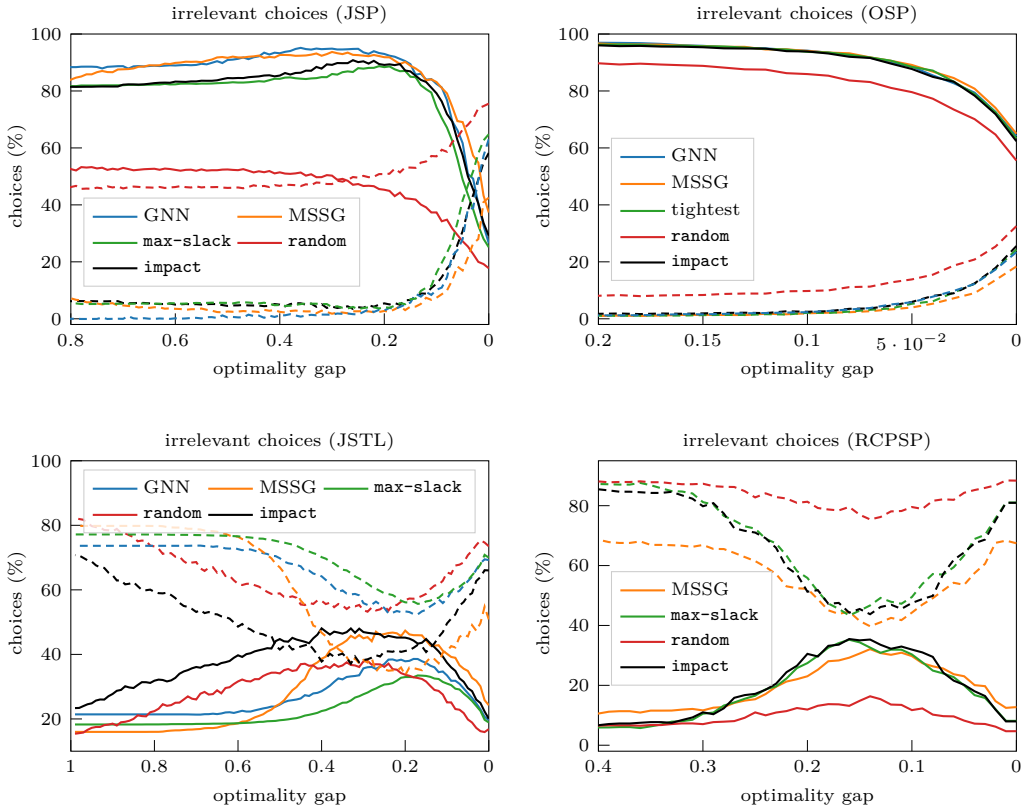

Figure 7 Mean percentage of irrelevant choices distinguished by category with no-good-learning. Bold lines represent the percentage of ICS's, dashed lines represent the percentage of ICUs. These results need to be interpreted with care since the procedure detailed in Section 2.2 is not exact in this case.

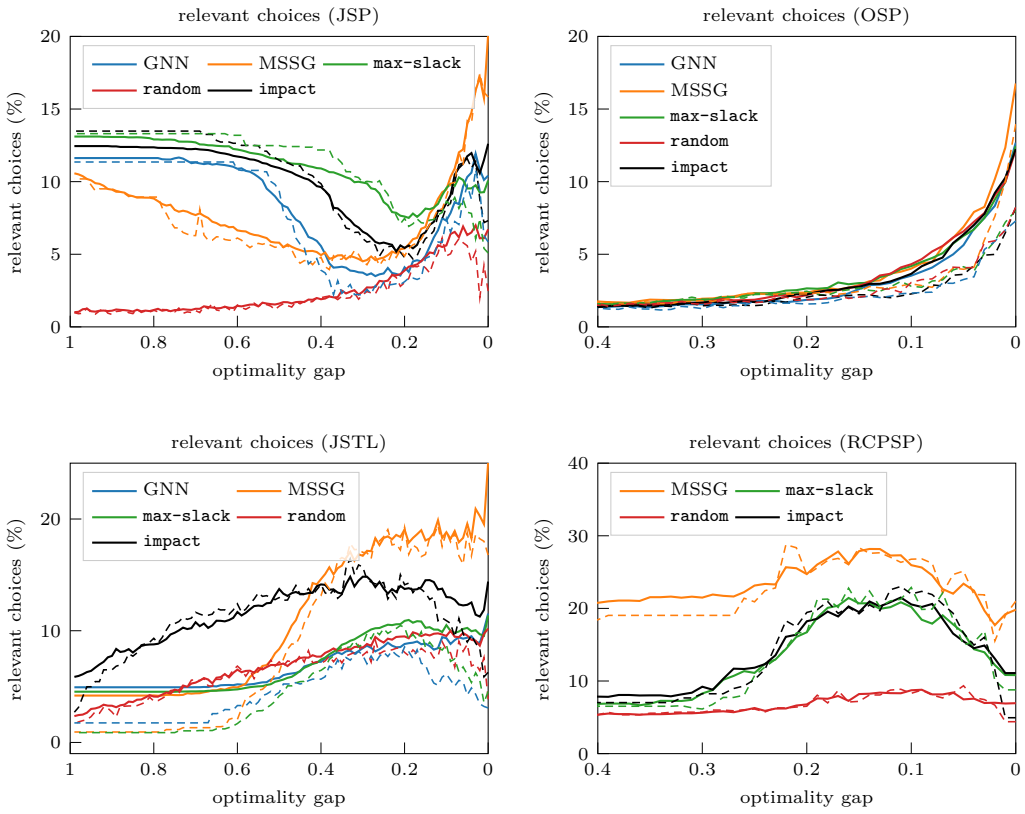

Figure 8 Percentage of relevant choices with no-good-learning. Bold lines represent mean values, dashed lines median values. These results need to be interpreted with care since the procedure detailed in Section 2.2 is not exact in this case.

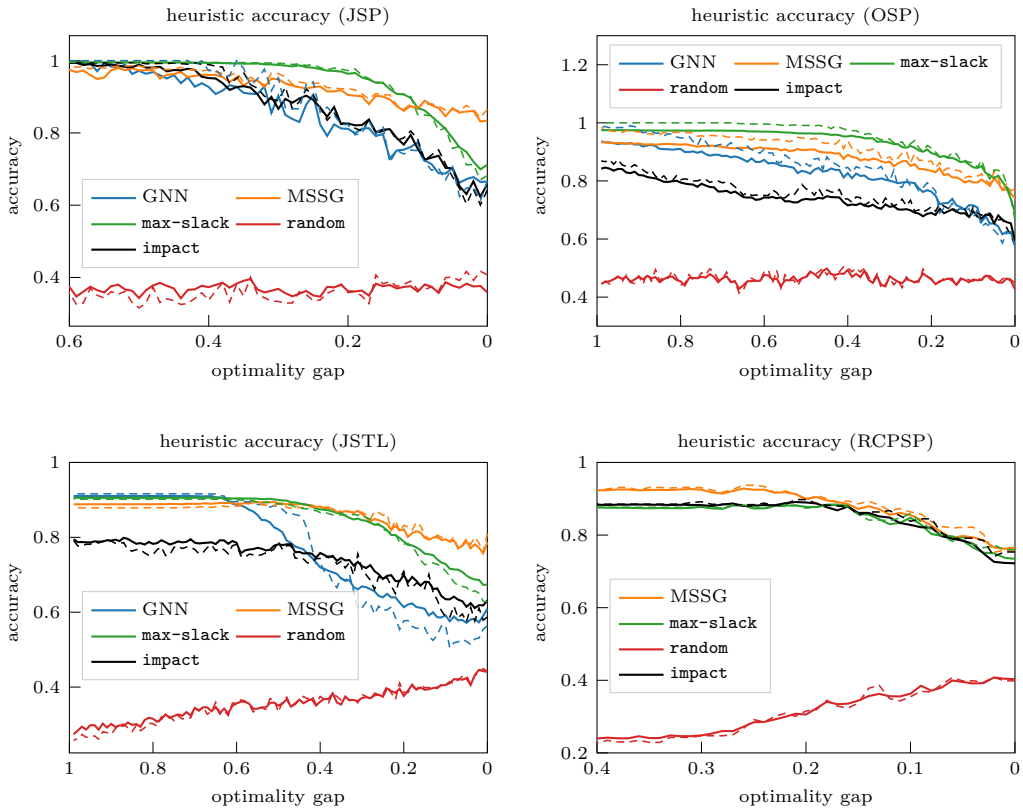

Figure 9 Online accuracy results with no-good-learning. These results need to be interpreted with care since the procedure detailed in Section 2.2 is not exact in this case.

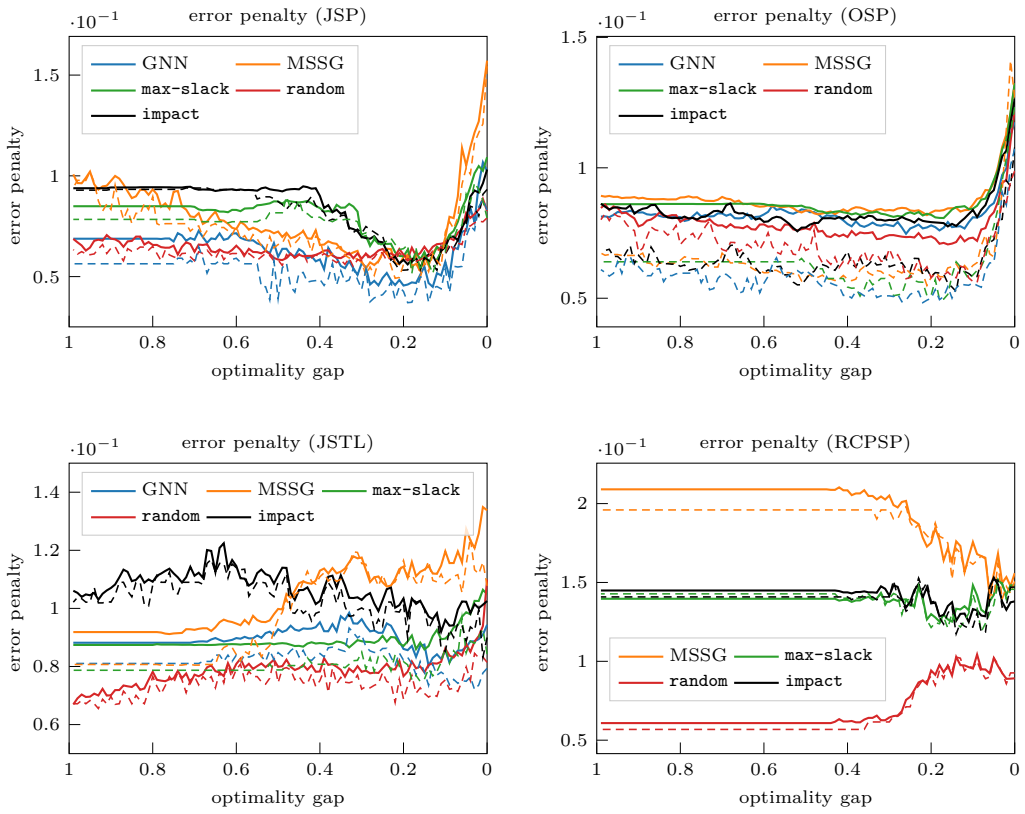

Figure 10 Error penalty results with no-good-learning. Bold lines represent mean values, dashed lines median values. These results need to be interpreted with care since the procedure detailed in Section 2.2 is not exact in this case.

Table 1 Instances used in all experiments.

instance type	size (# tasks)	instance names
JSP	50-500	abz5.txt, abz6.txt, abz8.txt, ft20.txt, la03.txt, la04.txt, la05.txt, la06.txt, la11.txt, la16.txt, la18.txt, la19.txt, la23.txt, la25.txt, orb03.txt, orb04.txt, orb05.txt, orb06.txt, orb07.txt, swv19.txt, swv20.txt, ta02.txt, ta05.txt, yn01.txt
OSP	9-225	GP03-01.txt, GP03-02.txt, GP04-04.txt, GP04-08.txt, GP05-04.txt, GP05-10.txt, GP06-02.txt, GP06-10.txt, GP07-02.txt, GP07-03.txt, GP08-02.txt, GP08-06.txt, GP09-03.txt, GP09-04.txt, GP10-08.txt, GP10-09.txt, j3-per0-2.txt, j3-per10-1.txt, j3-per20-0.txt, j4-per0-1.txt, j4-per20-0.txt, j4-per20-2.txt, j5-per0-1.txt, j5-per20-0.txt, j5-per20-1.txt, j6-per0-0.txt, j6-per10-1.txt, j6-per20-0.txt, j6-per20-1.txt, j7-per0-0.txt, j7-per0-1.txt, j7-per20-0.txt, j8-per0-1.txt, j8-per20-1.txt, j8-per20-2.txt, tai04_04_02.txt, tai04_04_04.txt, tai04_04_06.txt, tai05_05_04.txt, tai05_05_07.txt, tai05_05_08.txt, tai07_07_03.txt, tai07_07_05.txt, tai07_07_06.txt, tai10_10_02.txt, tai10_10_06.txt, tai10_10_07.txt, tai15_15_01.txt
JSTL	36-150	car5_0_0, car5_0_2, car6_0_0,5, car6_0_10, car7_0_10, car7_0_5, car8_0_10, car8_0_inf, ft06_0_1, ft06_0_inf, ft10_0_0, la01_0_0, la01_0_5, la02_0_0, la02_0_3, la03_0_1, la03_0_3, la04_0_0,5, la04_0_1, la05_0_0, 25, la05_0_3, la07_0_0,5, la07_0_10, la08_0_0,5, la08_0_2, la09_0_1, la09_0_10, la10_0_1, la11_0_10, la11_0_inf, la12_0_0, la12_0_1, la13_0_0, la13_0_10, la14_0_0,5, la14_0_1, la15_0_0,5, la15_0_inf, la16_0_0,5, la16_0_inf, la17_0_10, la17_0_3, la18_0_10, la18_0_3, la19_0_1, la19_0_inf, la20_0_1, la20_0_inf, la21_0_0,5, la21_0_10, la22_0_1, la22_0_10, la23_0_0, la23_0_10, la24_0_0,5, la24_0_10, la25_0_0, la25_0_10
RCPSP	30	j309_1.sm, j309_2.sm, j309_3.sm, j309_4.sm, j309_5.sm, j309_6.sm, j309_7.sm, j309_8.sm, j309_9.sm, j309_10.sm, j3013_1.sm, j3013_2.sm, j3013_3.sm, j3013_4.sm, j3013_5.sm, j3013_6.sm, j3013_7.sm, j3013_8.sm, j3013_9.sm, j3013_10.sm