# Simple Rules for Low-Knowledge Algorithm Selection*

J. Christopher Beck and Eugene C. Freuder

Cork Constraint Computation Centre, Department of Computer Science,
University College Cork, Cork, Ireland
{c.beck,e.freuder}@4c.ucc.ie

**Abstract.** This paper addresses the question of selecting an algorithm from a predefined set that will have the best performance on a scheduling problem instance. Our goal is to reduce the expertise needed to apply constraint technology. Therefore, we investigate simple rules that make predictions based on limited problem instance knowledge. Our results indicate that it is possible to achieve superior performance over choosing the algorithm that performs best on average on the problem set. The results hold over a variety of different run lengths and on different types of scheduling problems and algorithms. We argue that low-knowledge approaches are important in reducing expertise required to exploit optimization technology.

## 1 Introduction

Using constraint technology still requires significant expertise. A critical area of research if we are to achieve large scale adoption is the reduction of the skill required to use the technology. In this paper, we adopt a low-knowledge approach to automating algorithm selection for scheduling problems. Specifically, given an overall time limit $T$ to find the best solution possible to a problem instance, we run a set of algorithms during a short "prediction" phase. Based on the quality of the solutions returned by each algorithm, we choose one of the algorithms to run for the remainder of $T$. A low-knowledge approach is important in actually reducing the expertise required rather than simply shifting it to another portion of the algorithm selection process. Empirical analysis on two types of scheduling problems, disjoint algorithm sets, and a range of time limits demonstrates that such an approach consistently achieves performance no worse than choosing the best pure algorithm and furthermore can achieve performance significantly better.

The contributions of this paper are the introduction of a low-knowledge approach to algorithm selection, the demonstration that such an approach can achieve performance better than the best pure algorithm, and the analysis of the empirical results to characterize limits on the performance of any on-line algorithm selection technique.

* This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075 and ILOG, SA. Copyright ©Springer-Verlag

## 2 The Algorithm Selection Problem

The algorithm selection problem consists of choosing the best algorithm from a predefined set to run on a problem instance [1]. In AI, the algorithm selection problem has been addressed by building detailed, high-knowledge models of the performance of algorithms on specific types of problems. Such models are generally limited to the problem classes and algorithms for which they were developed. For example, Leyton-Brown et al. [2] have developed strong selection techniques for combinatorial auction algorithms that take into account 35 problem features based on four different representations. Other work applying machine learning techniques to algorithm generation [3] and algorithm parameterization [4–6] is also knowledge intensive, developing models specialized for particular problems and/or search algorithms and algorithm components.

Our motivation for this work is to lessen the expertise necessary to use optimization technology. While existing algorithm selection techniques have shown impressive results, their knowledge-intensive nature means that domain and algorithm expertise is necessary to develop the models. The overall requirement for expertise has not been reduced: it has been shifted from algorithm selection to predictive model building. It could still be argued that the expertise will have been reduced if the predictive model can be applied to different types of problems. Unfortunately, so far, the performance of a predictive model tends to be inversely proportional to its generality: while models accounting for over 99% of the variance in search cost exist, they are not only algorithm and problem specific, but also problem instance specific [7]. While the model building *approach* is general, the requirement for expertise remains: an in-depth study of the domain and of different problem representations is necessary to identify features that are predictive of algorithm performance. To avoid shifting the expertise to model building, we examine models that require no expertise to build. The feature used for prediction is the solution quality over a short period of time.

The distinction between low- and high-knowledge (or knowledge-intensive) approaches focuses on the number, specificity, and computational complexity of the measurements of a problem instance required to build a model. A low-knowledge approach has very few, inexpensive metrics, applicable to a very wide range of algorithms and problem types. A high-knowledge approach has more metrics, that are more expensive to compute, and are more specific to particular problems and algorithms. This distinction is independent of the model building approach. In particular, sophisticated model building techniques based on machine learning techniques are consistent with low-knowledge approaches.

## 3 On-line Scenario and Prediction Techniques

We use the following on-line scenario: a problem instance is presented to a scheduling system and that system has a fixed CPU time of $T$ seconds to return a solution. We assume that the system designer has been given a learning set of problem instances at implementation time and that these instances are representative of the problems that will be later presented. We assume that there

exists a set of algorithms, $A$, that can be applied to the problems in question. Algorithm selection may be done off-line by, for example, using the learning set to identify the best pure algorithm overall and running that on each problem instance. Alternatively, algorithm selection can be done on-line, choosing the algorithm only after the problem instance is presented. In the latter case, the time to make the selection must be taken into account. To quantify this, let $t_p$ represent the prediction time and $t_r$ the subsequent time allocated to run the chosen pure technique. It is required that $T = t_p + t_r$.

For the low-knowledge techniques investigated here, each pure algorithm, $a \in A$, is run for a fixed number of CPU seconds, $t$, on the problem instance. The results of each run are then used to select the algorithm that will achieve the best performance given the time remaining. We require that $t_p = |A| \times t$. The learning set is used to identify $t^*$ which is the value of $t$ that leads to the best system performance.

Three simple prediction rules each with three variations are investigated:

- **pcost** - Selection is based on the cost of the best solution found by each algorithm. The three variations are: $pcost\_min(t)$: the algorithm that has found the minimum cost solution over all algorithms by time $t$ is selected; $pcost\_mean(t)$: the algorithm with the minimum mean of the best solutions (sampled at 10 second intervals) is selected; $pcost\_median(t)$: identical to $pcost\_mean(t)$ except the median is used in place of the mean.
- **pslope** - Selection is based on the change in the cost of the best solutions found at 10 second intervals. The three variations are: $pslope\_min(t)$: the algorithm that has the minimum slope between $t-10$ and $t$ seconds is selected; $pslope\_mean(t)$: the algorithm with minimum mean slope for each pair of consecutive 10 second intervals is selected; $pslope\_median(t)$: identical to $pslope\_mean(t)$ except the median is used in place of the mean.
- **pextrap** - Selection is based on the extrapolation of the current cost and slope to a predicted cost at $T$. As above, the three variations are: $pextrap\_min(t)$: the best solutions for an algorithm at time $t$ and $t-10$ are used to define a line which is used to extrapolate the cost at time $T$; the algorithm that has the minimum extrapolated cost is chosen; $pextrap\_mean(t)$ the algorithm with the minimum mean extrapolated cost over each interval of 10 seconds from 20 seconds to $t$ seconds is selected; $pextrap\_median(t)$ identical to $pextrap\_mean(t)$ except the median is used in place of the mean.

For all rules ties are broken by selecting the algorithm with the best mean solution quality on the learning set at time $T$. The sampling interval was arbitrarily set to 10 seconds as it allowed time for a reasonable amount of search.

## 4 Initial Experiment

Our initial experiment divides a set of problem instances into a learning set and a test set, uses the learning set to identify $t^*$ for each prediction rule and variation, and then applies each rule variation using $t_p = |A| \times t^*$ to the test problems.

### 4.1 Problem Sets and Algorithms

Three sets of $20 \times 20$ job shop scheduling (JSP) problems are used. A total of 100 problem instances in each set were generated and 20 problems per set were chosen as the learning set. The rest were placed in the test set. The problem sets have different structure based on the generation of the activity durations.

- **Rand**: Durations are drawn randomly with uniform probability from the interval [1, 99].
- **MC**: Durations are drawn randomly from a normal distribution. The distributions for activities on different machines are independent. The durations are, therefore, machine-correlated (MC).
- **JC**: Durations are drawn randomly from a normal distribution. The distributions for different jobs are independent. Analogously to the MC set, these problems are job-correlated (JC).

These different problem structures have been studied for flow-shop scheduling [8] but not for job shop scheduling. They were chosen based on the intuition that the different structures may differentially favor one pure algorithm and therefore the algorithms would exhibit different relative performance on the different sets. Such a variation is necessary for on-line prediction to be useful: if one algorithm dominates on all problems, the off-line selection of that algorithm will be optimal.

Three pure algorithms are used. These were chosen out of a set of eight algorithms because they have generally comparable behavior on the learning set. The other techniques investigated performed much worse (sometimes by an order of magnitude) on every problem. The three algorithms are:

- **tabu-tsab**: a sophisticated tabu search due to Nowicki & Smutnicki [9]. The neighborhood is based on swapping pairs of adjacent activities on a subset of a randomly selected critical path. An important aspect of tabu-tsab is the use of an evolving set of the five best solutions found. Search returns to one of these solutions and moves in a different direction after a fixed number (1000 in our experiments) of iterations without improvement.
- **texture**: a constructive search technique using texture-based heuristics [10], strong constraint propagation [11, 12], and bounded chronological backtracking. The bound on the backtracking follows the optimal, zero-knowledge pattern of 1, 1, 2, 1, 1, 2, 4, ... [13]. The texture-based heuristic identifies a resource and time point with maximum competition among the the activities and chooses a pair of unordered activities, branching on the two possible orders. The heuristic is randomized by specifying that the resource and time point is chosen with uniform probability from the top 10% most critical resources and time points.
- **settimes**: a constructive search technique using the SetTimes heuristic [14], the same propagation as texture, and slice-based search [15], a type of discrepancy-based search. The heuristic chronologically builds a schedule by examining all activities with minimal start time, breaking ties with minimal end time, and then breaking further ties arbitrarily. The discrepancy bound follows the pattern: 2, 4, 6, 8, ....

### 4.2 Experimental Details

For these experiments, the overall time limit, $T$, is 1200 CPU seconds. Each pure algorithm is run for $T$ seconds with results being logged whenever a better solution is found. This design lets us process the results to examine the effect of different settings for the prediction time, $t_p$, and different values for $T \leq 1200$. As noted, the number of algorithms, $|A|$, is 3.

To evaluate the prediction rules, we process the data as follows. Given a pure algorithm time of $t = t_p/|A|$ seconds, we examine the best makespans found by each algorithm on problem instance $k$ up to $t$ seconds. Based on the prediction rule, one algorithm, $a^*$, is chosen. We then examine the best makespan found by $a^*$ for $k$ at $t + t_r$ where $t_r = T - t \times |A|$. This evaluation means that we are assuming that each pure algorithm can be run for $t$ CPU seconds, one can be chosen, and that chosen one can continue from where it left off.

### 4.3 Results

**Learning Set** Table 1 displays the fraction of learning problems in each subset and overall for which each algorithm found the best solution. It also shows the mean relative error (MRE), a measure of the mean extent to which an algorithm finds solutions worse than the best known solutions. MRE is defined as follows:

$$MRE(a, K) = \frac{\sum_{k \in K} \frac{c(a,k) - c^*(k)}{c^*(k)}}{|K|} \tag{1}$$

Where:

- $K$ is a set of problem instances
- $c(a, k)$ is the lowest cost solution found by algorithm $a$ on $k$
- $c^*(k)$ is the lowest cost solution known $k$.

Tabu-tsab finds the best solution for slightly more problems than texture and produces the lowest MRE. These differences are slight as in 50% of the problems, texture finds the best solution. As expected, there are significant differences among the problems sets: while tabu-tsab clearly dominates in the MC problem set, the results are more uniform for the random problem set and texture is clearly superior in the JC set.

| | MC | | Rand | | JC | | All | |
|---|---|---|---|---|---|---|---|---|
| | Frac. Best | MRE | Frac. Best | MRE | Frac. Best | MRE | Best | MRE |
| tabu-tsab | 0.7 | 0.00365 | 0.6 | 0.00459 | 0.3 | 0.00688 | 0.53 | 0.00504 |
| texture | 0.2 | 0.01504 | 0.4 | 0.00779 | 0.9 | 0.00092 | 0.5 | 0.00792 |
| settimes | 0.1 | 0.03752 | 0 | 0.03681 | 0.5 | 0.00826 | 0.2 | 0.02753 |

**Table 1.** Fraction of problems in each learning problem set for which the best solution was found by each algorithm (Frac. Best) and their mean relative error (MRE).

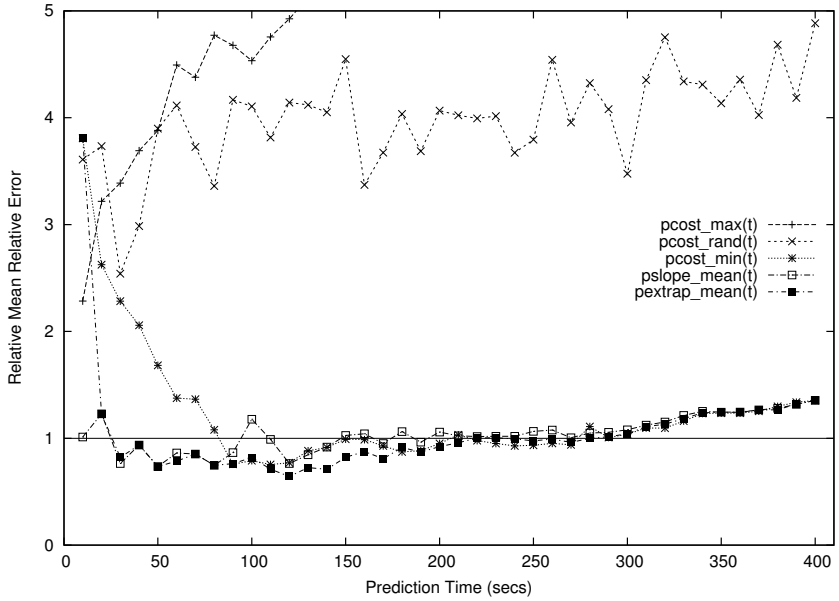


**Fig. 1.** The performance of the best variation of each prediction rule at prediction time $t \in \{10, 20, \ldots, 400\}$ on the JSP learning set. The graph shows the MRE relative to the MRE achieved by the best pure algorithm, tabu-tsab.

The best variation of each prediction rule are shown in Figure 1. This graph presents the relative MRE (RMRE) found by the prediction rule as we varied the prediction time, $t \in \{10, 20, \ldots, 400\}$. The RMRE displayed for each prediction rule, $p$, is the MRE relative to the MRE of best pure algorithm, in this case tabu-tsab, calculated as follows: $RMRE(p(t)) = MRE(p(t))/MRE(\textit{tabu-tsab})$. Values below $y = 1$ represent performance better than the best pure algorithm. For example, the RMRE for $\textit{pextrap\_mean}(50)$ is 0.736: the MRE achieved by the *pextrap_mean* rule at $t = 50$ is 73.6% that achieved by tabu-tsab.

It is possible that the pure algorithms have such similar performance that any prediction rule would perform well. Therefore, two additional "straw men" prediction rules are included in Figure 1: $\textit{pcost\_max}(t)$ and $\textit{pcost\_rand}(t)$. The algorithm whose best solution at $t$ is the maximum over all algorithms is chosen in the former and a random algorithm is chosen in the latter. These two techniques perform substantially worse than the real prediction rules, lending support to the claim that the observed results are not due to a floor effect.

The best performance for each prediction rule is seen with $\textit{pcost\_min}(110)$, $\textit{pslope\_mean}(50)$, and $\textit{pextrap\_mean}(120)$. The differences between the MRE of each prediction rule and tabu-tsab are not statistically significant.[1]

[1] All statistical results in this paper are measured using a randomized paired-$t$ test [16] and a significance level of $p \leq 0.005$.

**Test Set** Table 2 displays the fraction of the problems in the test set for which each algorithm found the best solution (Fraction Best) and the MRE for each pure algorithm and for the best variation and prediction time, $t^*$, of each prediction rule. On the basis of the fraction of best solutions, all prediction rules are worse than the best pure algorithm (texture) however none of these differences are statistically significant. Based on MRE, while tabu-tsab and texture are very closely matched, settimes performs significantly worse and each prediction rule performs better than each pure algorithm. Statistically, however, only $pcost\_min(110)$ achieves performance that is significantly better than the best pure algorithm. In fact, $pcost\_min(t)$ is robust to changes in $t$ as a difference at the same level of significance is found for all $t \in \{80, \ldots, 260\}$.

| Algorithm | $t^*$ | Fraction Best | MRE |
|---|---|---|---|
| tabu-tsab | - | 0.5125 | 0.00790 |
| texture | - | 0.5458 | 0.00859 |
| settimes | - | 0.125 | 0.02776 |
| $pcost\_min$ | 110 | 0.5292 | **0.00474*** |
| $pslope\_mean$ | 50 | 0.5208 | 0.00726 |
| $pextrap\_mean$ | 120 | 0.475 | 0.00577 |
| static | - | **0.725*‡** | **0.00460*** |

**Table 2.** The performance of each pure algorithm and the prediction techniques on the test set. '*' indicates that the prediction technique achieved an MRE significantly lower or found the best solution in a significantly higher fraction of problem instances than the best pure algorithm. '‡' indicates that the static prediction technique found the best solution in a significantly greater fraction of problems than $pcost\_min$.

**A Static Prediction Technique** The existence of widely differing pure algorithm performance on the different problem subsets (Table 1) suggests that a high-knowledge, static prediction technique could be built based on categorizing a problem instance into one of the subsets and then using the algorithm that performed best on that subset in the learning phase. The *static* prediction technique uses texture on the JC problems and tabu-tsab on the other two sets. The results for static presented in Table 2 make two strong assumptions: the mapping of a problem instance to a subset is both infallible and takes no CPU time. These assumptions both favor the static technique over the low-knowledge prediction techniques. The results indicate that the static technique outperforms all the other prediction techniques and the pure algorithms in terms of the fraction of problems solved and does the same as $pcost\_min$ on MRE.

The static technique is knowledge-intensive: one has to know to look for the specific duration structure before algorithm performance correlations can be developed. Therefore, we are not interested specifically in the static technique. It is included to demonstrate that a high-knowledge technique, even under idealized assumptions, may not significantly out-perform a low-knowledge technique.

## 5 Investigations of Generality

Our initial experiment demonstrates that, at least for the problem sets, algorithms, and time limit used, it is possible to use low-knowledge prediction and simple rules to do algorithm selection. Furthermore $pcost\_min(t)$ achieves MRE performance that is significantly better than the best pure algorithm and comparable to an idealized high-knowledge approach. A number of questions are raised with respect to the generality of these results. How sensitive are the results to different choices of parameters such as the overall time limit? Can such simple rules be successfully applied to other problems and algorithms? Can we develop a characterization of the situations in which such methods are likely to be successful? Can we evaluate the results of the prediction rules in an absolute sense and therefore provide intuitions as to the likelihood that more sophisticated prediction techniques may be able to improve upon them? In this section, we will address these questions.

### 5.1 Other Time Limits

In all experiments presented above, the overall CPU time limit, $T$, was 1200 seconds. Table 3 reports a series of experiments with $T \in \{100, 200, \ldots, 1200\}$. For each time limit, we repeated the experiment: $t^*$, the prediction time with the lowest MRE on the learning set for the best variation of each prediction rule, was identified, each problem in the test set was solved with each prediction rule using its $t^*$ value, and the MRE was compared against the best pure algorithm. There were no significant differences between the MRE of the best pure technique and those of the prediction rules across all the $T$ values on the learning set. The results for the test set are displayed in the final four columns. For time limits $500 \leq T \leq 1200$, $pcost\_min(t^*)$ performs significantly better than the best pure technique. For $T = 100$ the best pure technique (texture) has a significantly lower MRE than $pcost\_min(t^*)$ and $pslope\_min(t^*)$. For $T = 100$, the static technique is able to find significantly lower RMREs than $pcost\_min$. No other time limits showed any difference between static and $pcost\_min$. These results indicate that the results using $T = 1200$ are relatively robust to different $T$ values.

### 5.2 Other Problems

Earliness/tardiness scheduling problems (ETSPs) define a set of jobs to be scheduled on a set of resources such that each job has an associated due date and costs associated with finishing the last activity in a job before or after that due date. The activities within jobs are completely ordered and the resources can only execute a single activity at any time. Three ETSP algorithms are used here:

- **hls**: a hybrid local search algorithm combining tabu search with linear programming.
- **mip**: a pure mixed-integer programming approach using the default search heuristics in CPLEX 7.2 with an emphasis on good solutions over optimal.

| Time Limit | Learning Set | | | | | | Test Set | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | pcost | | pslope | | pextrap | | pcost | pslope | pextrap | static |
| | RMRE | $t^*$ | RMRE | $t^*$ | RMRE | $t^*$ | RMRE | | | |
| 100 | 1.114 | 20 | 1.139 | 20 | 1.094 | 20 | **1.110†** | **1.134†** | 1.062 | **0.821*‡** |
| 200 | 1.138 | 40 | 1.004 | 30 | 0.992 | 30 | 1.001 | 1.103 | 0.990 | **0.793*** |
| 300 | 1.048 | 60 | 0.967 | 30 | 0.935 | 30 | 0.925 | 1.067 | 1.012 | **0.782*** |
| 400 | 0.969 | 90 | 0.879 | 30 | 0.895 | 50 | 0.914 | 1.039 | 0.912 | **0.783*** |
| 500 | 0.849 | 90 | 0.761 | 30 | 0.730 | 50 | **0.814*** | 1.043 | 0.920 | **0.776*** |
| 600 | 0.815 | 110 | 0.751 | 50 | 0.740 | 50 | **0.799*** | 1.024 | 0.964 | **0.738*** |
| 700 | 0.824 | 110 | 0.683 | 50 | 0.683 | 50 | **0.752*** | 0.926 | 0.882 | **0.707*** |
| 800 | 0.772 | 100 | 0.668 | 50 | 0.668 | 50 | **0.683*** | 0.921 | 0.877 | **0.689*** |
| 900 | 0.748 | 110 | 0.702 | 30 | 0.679 | 120 | **0.650*** | 0.977 | 0.772 | **0.660 *** |
| 1000 | 0.681 | 90 | 0.663 | 50 | 0.646 | 120 | **0.625*** | 0.878 | 0.635 | **0.633*** |
| 1100 | 0.680 | 90 | 0.671 | 50 | 0.600 | 120 | **0.630*** | 0.909 | 0.724 | **0.618*** |
| 1200 | 0.754 | 110 | 0.736 | 50 | 0.642 | 120 | **0.600*** | 0.919 | 0.730 | **0.583*** |

**Table 3.** The results of the best variations of the prediction rules relative to the best pure technique for different run-time limits for the JSP problems. '*' indicates that the prediction rule achieved an RMRE significantly lower than the best pure algorithm, '†' indicates that the best pure technique is significantly better than the prediction rule, and '‡' indicates a time limit where static is significantly better than *pcost_min*.

– **probeplus**: a probe-based algorithm combining linear programming and constraint programming search.

Details of these algorithms, problems sets, and results can be found in Beck & Refalo [17].

We divided the 90 ETSP problems into a learning set of 36 problems and a test set of 54 problems. The experimental design is identical to our first experiment. In particular, the overall time, $T = 1200$, and the number of algorithms, $|A| = 3$.

Instead of makespan minimization, the optimization criteria on ETSPs is the minimization of weighted earliness/tardiness cost. It is possible for problems to have a optimal cost of 0 and a number of the easier problem instances do. Therefore, MRE is not well-formed as it would require a division by 0. Instead, we calculate the normalized cost (NC) for each problem and use the mean normalized cost (MNC) as one of our evaluation criteria. NC is commonly used in work that has applied genetic algorithms to ETSPs [18]. In that literature, the cost of a solution is divided by the sum of the durations of all activities in the problem weighted by the earliness/tardiness cost of each job. In our problems, the earliness and tardiness weights for a single job are independent. Therefore, we have modified this normalization to weight the duration sum with the mean of the two cost weights. The NC for algorithm $a$ on problem instance $k$ is

$$NC(a,k) = \frac{c(a,k)}{\sum_{j \in Jobs(k)} (\frac{ec_j + tc_j}{2} \times \sum_{a \in Job_j} dur_a)} \tag{2}$$

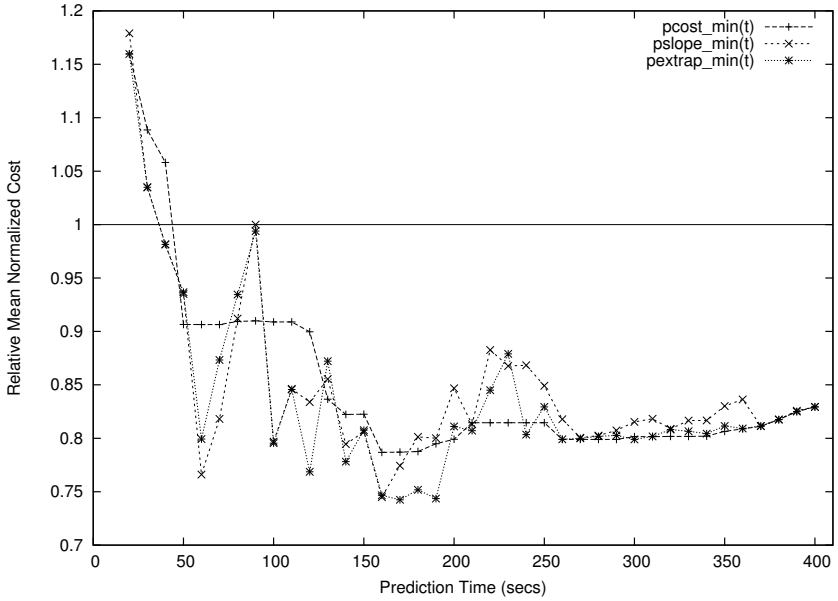


**Fig. 2.** The performance of the best variations of the three prediction rules at different prediction times on the ETSP learning set. The graph plots the mean normalized costs of each rule at each $t$ value relative to the mean normalized cost achieved by the best pure algorithm.

Where:

- $c(a, k)$ is the lowest cost for algorithm $a$ on problem instance $k$
- $Jobs(k)$ is the set of jobs in problem instance $k$
- $Job_j$ is the set of activities in job $j$
- $ec_j$ and $tc_j$ are respectively the earliness and tardiness costs for job $j$

Figure 2 presents the MNC of the three best prediction rule variations (relative to the best pure technique, hls for the learning set) with $t \in \{20, 30, \ldots, 400\}$. The plot is analogous to Figure 1. For each prediction rule the "min" variations results in the best performance with the following $t^*$ values: $pcost\_min(160)$, $pslope\_min(160)$, and $pextrap\_min(170)$. As with the JSP problem set, however, none of these results are significantly different from those found by hls on the learning set.

Table 4 presents the fraction of the test problems for which each pure and prediction-based technique found the best solution and the MNC over all problem instances in the test set. The prediction rules perform very well on both measures. However, none of them achieve performance on either measure that is significantly different from the best pure technique. The pure technique that achieves the best solution on the highest number of problem instances (hls) is worst on the basis of MNC. The reverse is also true, as mip finds the lowest MNC but finds the best solution on the fewest number of instances.

| Algorithm | $t^*$ | Fraction Best | MNC |
|---|---|---|---|
| mip | - | 0.4259 | 0.02571 |
| hls | - | 0.6481 | 0.02908 |
| probeplus | - | 0.5 | 0.02645 |
| pcost_min | 160 | 0.6296 | 0.01721 |
| pslope_min | 160 | 0.7407 | 0.01753 |
| pextrap_min | 170 | 0.6667 | 0.01825 |

**Table 4.** The mean normalized cost (MNC) for each pure technique and the best prediction rules on the ETSP test set. None of the prediction rules achieve a significantly different MNC or fraction best than the best pure technique.

### 5.3 Characterizations of Prediction Techniques

Clearly, two interacting factors determine the performance of the prediction rules tested in this paper and, indeed, any on-line prediction technique: the accuracy of prediction and the computation time required to make the prediction.

We expect prediction accuracy to increase as $t_p$ is increased since more computation time will result in better data regarding algorithm performance. Furthermore, since we have a fixed time limit, the larger $t_p$, the closer it is to this time limit and the less far into the future we are required to predict. To evaluate the data underlying the accuracy of predictions for the *pcost* rule, in Figure 3 we present the mean Spearman's rank correlation coefficient between $t$ and $t + t_r$ for the learning sets of both the JSP and the ETSP problems. For a problem instance, $k$, and prediction time, $t$, we rank each of the pure algorithms in ascending order of the best makespan found by time $t$. We then create the same ranking at time $t + t_r$, the total run-time of the chosen algorithm. The correlation between these rankings is calculated using Spearman's rank correlation coefficient and the mean coefficient over all the problems in the set is plotted. It is reasonable to expect that the accuracy of $pcost\_min(t)$ depends on the extent to which the algorithm ranking at time $t$ is correlated with that at $t + t_r$. We can see in the graph that the lower the value of $t$, the lower the correlation and, therefore, the lower the accuracy of the predictions. Both from the graph and from the reasoning above, to achieve a greater accuracy, prediction should be as late as possible.

For $t = 10$ the JSP rankings are negatively correlated. The appropriate heuristic for choosing a pure algorithm at $t = 10$ is to choose the algorithm whose best makespan is *largest*. This is exactly $pcost\_max(t)$ plotted in Figure 1 and in that graph, $pcost\_max(10)$ does indeed perform better than $pcost\_min(10)$.

The second factor is the time required to measure the instance and make the prediction. In an on-line context, more time spent predicting means less spent solving the problem with the chosen algorithm. If $T = 1200$ and $|A| = 3$, then $t = 200$ means that 600 seconds have expired when the algorithm choice is made. Only 600 additional seconds are available to run the chosen algorithm. This has a large implication for performance of prediction-based techniques. This is illustrated in Figure 4. The $pperf(t)$ plot is the MRE of a perfect prediction

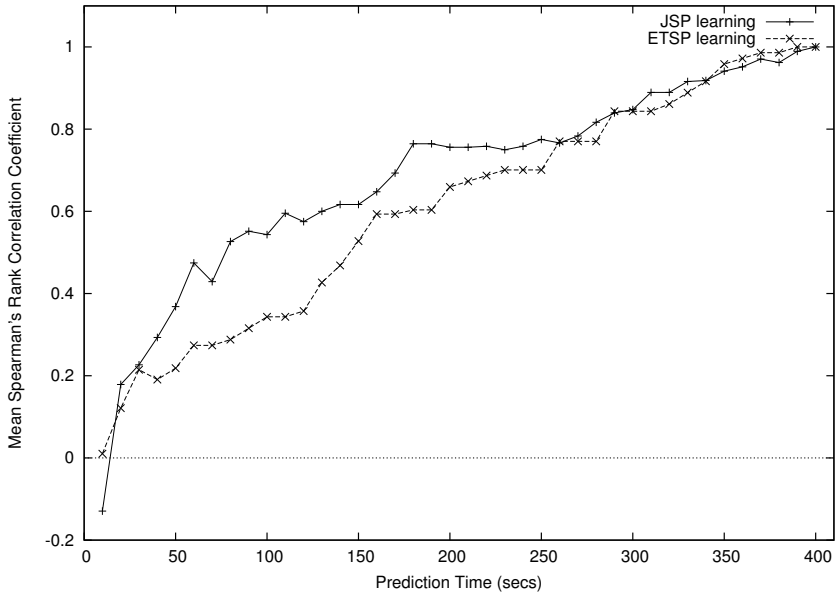


**Fig. 3.** The mean Spearman's rank correlation coefficient between rankings of the pure algorithms at prediction time, $t$, and $t + t_r$ for problems in the JSP learning set and the ETSP learning set.

on the test set. For example, for $t = 200$, the effective run time of the chosen technique is 800 seconds: 200 seconds during the prediction phase and then the remaining 600 seconds. The perfect MRE for $t = 200$ therefore is found using the lowest makespan found by any pure technique by time 800 and calculating the error compared to the best known makespan. When $t$ is very small, the MRE of $pperf(t)$ is very small too. This reflects the fact that the pure algorithms do not find large improvements extremely late in the run. As the $t$ increases however, the best case MRE increases: the time used in prediction instead of solving results in worse performance even with perfect prediction.

These graphs demonstrate the trade-off inherent for any on-line prediction technique: for accuracy the prediction time should be as late as possible but to allow time for the pure algorithm to run after it is chosen, it should be as early as possible. While the correlation graph presents data specific to a prediction rule used in this paper, we expect a similar graph of accuracy vs. the prediction time for all prediction techniques. The perfect prediction graphs clearly have a general interpretation since, by definition, no prediction technique can achieve better performance.

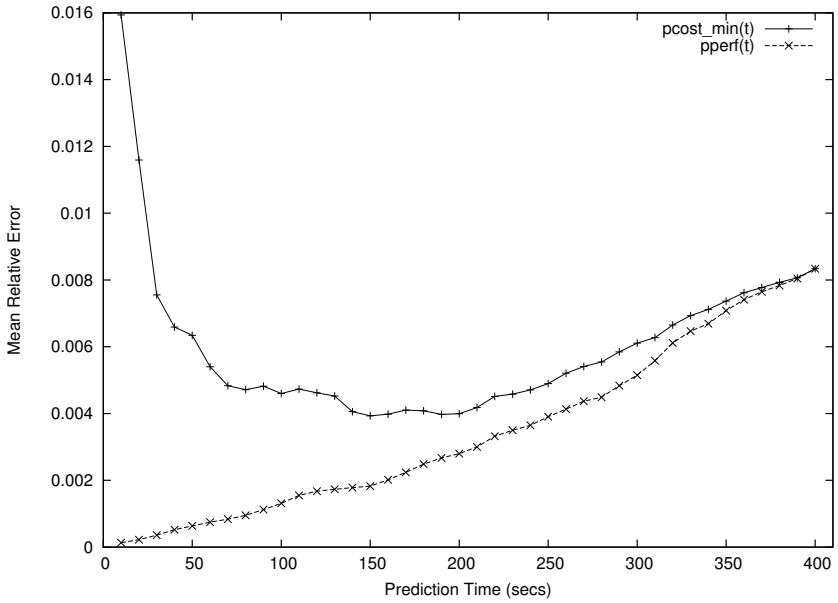


**Fig. 4.** The MRE on the JSP test set for $pcost\_min(t)$ and when we assume perfect prediction.

## 6 Discussion

We have shown that low-knowledge metrics of pure algorithm behavior can be used to form a system that performs as well, and sometimes better, than the best pure algorithm. If our goal was to win the algorithmic "track meet" and publish better results, our results are not spectacular. However, our goal was not to build a better algorithm through applying our expertise. Our goal was to exploit existing techniques with minimal expertise. From that perspective, the fact that applying simple rules to an instance-specific prediction phase is able to outperform the best pure algorithm is significant. We believe this study serves as a proof-of-concept of low-knowledge approaches and indicates that they are an important area of study in their own right.

Beyond the importance of low-knowledge approaches to reduce expertise, a prosaic reason to develop these approaches is that they can provide guidance in deciding whether the effort and expense of applying human expertise is worthwhile. Figure 3 shows that at a prediction time of $t = 100$ the mean $r$-value for $pcost\_min(t)$ on the JSP learning set is 0.543. This is a relatively low correlation, providing support for the idea that a more informed approach can significantly increase prediction accuracy. On the other hand, if we expected the on-line computation required for a high-knowledge approach to take more time (e.g., $t = 250$), the return on an investment in a high-knowledge approach seems less likely: the mean $r$-value is 0.775 so there is less room for improvement in pre-

diction accuracy. Similarly, Figure 4 shows that at a prediction time of $t = 100$ the MRE of *pcost_min* on the JSP test set is 0.0046. Based on the $pperf(t)$ plot, any predictive approach can only reduce this MRE to 0.0013. Is the development of a high-knowledge model worth the maximum theoretical reduction in MRE from 0.46% to 0.13%? In high cost domains (e.g., airline scheduling) such an effort would be worthwhile. In other domains (e.g., a manufacturing plant with uncertainty) such a difference is irrelevant. The results of easy to implement low-knowledge techniques can therefore guide the system development effort in the more efficient allocation of resources.

## 7 Future Work

We intend to pursue two areas of future work. The first, directly motivated by existing high-knowledge approaches, is the application of machine learning techniques to low-knowledge algorithm selection. The variety of features that these techniques can work with will be much more limited, but we expect that better grounded techniques can improve prediction accuracy and system performance over the simple rules. The second area for future work is to move from "one-shot" algorithm selection to on-line control of multiple algorithms. The decision making could be extended to allow the ability to dynamically switch among pure algorithms based on algorithm behavior.

Another consideration is the types of problems that are appropriate for prediction techniques or control-level reasoning. A real system is typically faced with a series of changing problems to solve: a scheduling problem gradually changes as new orders arrive and existing orders are completed. As the problem or algorithm characteristics change, prediction-based techniques may have the flexibility to appropriately change the pure algorithms that are applied.

## 8 Conclusion

We have shown that a low-knowledge approach based on simple rules can be used to select a pure algorithm for a given problem instance and that these rules can lead to performance that is as good, and sometimes better, than the best pure algorithm. We have argued that while we expect high-knowledge approaches will result in better performance, low-knowledge techniques are important from the perspective of reducing the expertise required to use optimization technology and have a useful role in guiding the expert in deciding when high-knowledge approaches are likely to be worthwhile.

## References

1. Rice, J.: The algorithm selection problem. Advances in Computers **15** (1976) 65–118

2. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP02). (2002) 556–572
3. Minton, S.: Automatically configuring constraint satisfaction programs: A case study. CONSTRAINTS **1** (1996) 7–43
4. Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B., Chickering, M.: A bayesian approach to tacking hard computational problems. In: Proceedings of the Seventeenth Conference on uncertainty and Artificial Intelligence (UAI-2001). (2001) 235–244
5. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic restart policies. In: Proceedings of the Eighteenth National Conference on Artifiical Intelligence (AAAI-02). (2002) 674–681
6. Ruan, Y., Horvitz, E., Kautz, H.: Restart policies with dependence among runs: A dynamic programming approach. In: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002), Springer-Verlag (2002) 573–586
7. Watson, J.P.: Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem. PhD thesis, Dept. of Computer Science, Colorado State University (2003)
8. Watson, J.P., Barbulescu, L., Whitley, L., Howe, A.: Constrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. INFORMS Journal on Computing **14** (2002)
9. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science **42** (1996) 797–813
10. Beck, J.C., Fox, M.S.: Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. Artificial Intelligence **117** (2000) 31–81
11. Nuijten, W.P.M.: Time and resource constrained scheduling: a constraint satisfaction approach. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology (1994)
12. Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artificial Intelligence **143** (2003) 151–188
13. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters **47** (1993) 173–180
14. Scheduler: ILOG Scheduler 5.2 User's Manual and Reference Manual. ILOG, S.A. (2001)
15. Beck, J.C., Perron, L.: Discrepancy-bounded depth first search. In: Proceedings of the Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CPAIOR'00). (2000)
16. Cohen, P.R.: Empirical Methods for Artificial Intelligence. The MIT Press, Cambridge, Mass. (1995)
17. Beck, J.C., Refalo, P.: Combining local search and linear programming to solve earliness/tardiness scheduling problems. In: Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'02). (2002)
18. Vazquez, M., Whitley, L.D.: A comparision of genetic algorithms for the dynamic job shop scheduling problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann (2000) 1011–1018