

Low-Knowledge Algorithm Control*

Tom Carchrae and J. Christopher Beck

Cork Constraint Computation Center
University College Cork, Ireland
{t.carchrae, c.beck}@4c.ucc.ie

Abstract

This paper addresses the question of allocating computational resources among a set of algorithms in order to achieve the best performance on a scheduling problem instance. Our primary motivation in addressing this problem is to reduce the expertise needed to apply constraint technology. Therefore, we investigate algorithm control techniques that make decision based only on observations of the improvement in solution quality achieved by each algorithm. We call our approach “low-knowledge” since it does not rely on complex prediction models. We show that such an approach results in a system that achieves significantly better performance than all of the pure algorithms without requiring additional human expertise. Furthermore the low knowledge approach achieves performance equivalent to a perfect high-knowledge classification approach.

Introduction

Despite both the commercial and academic success of optimization technology and specifically constraint programming, using the technology still requires significant expertise. For non-trivial applications the quality of a system still has much to do with the quality of the person that implemented it (Le Pape *et al.* 2002). In this paper, we investigate algorithm control techniques aimed at achieving strong scheduling performance using off-the-shelf algorithms without requiring significant human expertise. This is done through the application of machine learning techniques to “low-knowledge” algorithm control. Rather than building knowledge-intensive models relating algorithm performance to problem features, we base the control decisions on the evolution of solution quality over time. Such an approach is crucial to our goal of the reduction of expertise.

Given a time limit T to find the best solution possible to a problem instance, we investigate two control paradigms. The first, *predictive*, paradigm runs a set of algorithms during a prediction phase and chooses one to run for the remainder of T . A Bayesian classifier is trained on a set of learning instances and used to identify the length of the predic-

tion phase as well as the selected algorithm. In the second, *switching*, paradigm the control decisions allocate computational resources to each algorithm over a series of iterations such that the total run-time is T . Reinforcement learning is used during the solving to allocate the run-time.

The contributions of this paper are the introduction of a low-knowledge approach to algorithm control and the demonstration that such an approach can achieve performance significantly better than the best pure algorithm and as good as a perfect high-knowledge approach.

Low-Knowledge Algorithm Control

Our goal is to build a system that finds the best solution to an instance of a scheduling problem in the following 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 encountered later. We assume that there exists a set of algorithms, A , that can be applied to the problems in question. Algorithm control is the way that the pure algorithms, $a \in A$, are used to solve the problem instance.

Algorithm control and its more specific form, the algorithm selection problem (Rice 1976), have been addressed by building detailed, high-knowledge models of the performance of algorithms on specific problems types. Such models are generally limited to the problem classes and algorithms for which they were developed. For example, Leyton-Brown *et al.* (Leyton-Brown, Nudelman, & Shoham 2002) developed strong selection techniques for combinatorial auction algorithms that take into account 35 problem features and their interactions based on three problem representations. Lagoudakis & Littman (Lagoudakis & Littman 2000) use a detailed mathematical model of algorithm behavior, Markov decision processes, and reinforcement learning for algorithm control for sorting and order statistic selection tasks. Other work applying machine learning techniques to algorithm generation (Minton 1996) and algorithm parameterization (Horvitz *et al.* 2001; Kautz *et al.* 2002; Ruan, Horvitz, & Kautz 2002) is also knowledge intensive, developing models specialized for particular problems and/or search algorithms and algorithm components.

Our motivation for addressing algorithm control is to

*This work has received support from Science Foundation Ireland (Grant 00/PI.1/C075), Irish Research Council for Science, Engineering, and Technology (Grant SC/2003/82), and ILOG, SA. Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

lessen the expertise necessary to use optimization technology. While existing algorithm control techniques have shown impressive results, their knowledge-intensive nature means that domain and algorithm expertise is necessary to develop the models. The requirement for expertise has not, therefore, been reduced; it has been shifted from algorithm building 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, 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 problem instance specific (Watson 2003). The model building *approach* is general, but the requirement for expertise remains: an in-depth study of the domain and of different problem representations is necessary in order to identify a set of relevant features that are predictive of algorithm performance. To avoid shifting the expertise to model building, we examine approaches that require little expertise.

The distinction between low- and high-knowledge (or knowledge-intensive) approaches focuses on the number, specificity, and computational complexity of the measurements that need to be made of a problem instance. A low-knowledge approach has very few, inexpensive metrics, applicable to a wide range of algorithms and problem types. A high-knowledge approach has more metrics, that are more expensive to compute, and more specific to particular problems and algorithms. This distinction is independent of the approach taken to build a predictive model. In particular, sophisticated model building techniques (e.g., based on machine learning) are consistent with low-knowledge approaches provided the observations on which they are built do not require significant expertise to identify.

We make some basic assumptions about the pure algorithms for which our techniques are appropriate. First, after a brief start-up time (at most, a few seconds) an algorithm is always able to return the best, complete solution it has found so far. Second, we require that an algorithm is able to take an external solution and search for solutions that are better. In the latter case, if an algorithm has not found a better solution, when asked for its best solution, the algorithm returns the external solution.

A Bayesian Classifier for Prediction

Let t_p represent the prediction time and t_r the subsequent time allocated to run the chosen pure technique. It is required that $T = t_p + t_r$. During the prediction phase, each pure algorithm, $a \in A$, is run for a fixed number of CPU seconds, t , on the problem instance. The quality of solutions found for each run is used to select the algorithm that will achieve the best performance given the time remaining. We require that $t_p = |A| \times t$. We assume that when a pure algorithm has been selected, it does not have to re-start: it can continue the search from where it left off in the prediction phase. The total run-time for the selected technique is $T_s = t_r + t_p/|A|$.

A simple categorical Bayesian model is constructed to select the algorithm given a predefined t_p . For every 10 sec-

onds of prediction time and for each pure algorithm, a binary variable is created with possible values of “best” or “behind”. The value is assigned depending on whether the algorithm has found the best solution of all the algorithms, $a \in A$, at that time. Two or more algorithms may have both found the best solution at a particular time. For each algorithm, we also have a binary variable representing the final performance of the algorithm (i.e, at time T_s). It also has the value “best” or “behind” depending on the algorithm’s comparison with all the other algorithms. For the learning phase, these variables are input. For the test phase, the value of the final performance variables for each algorithm are predicted by the constructed Bayesian classifier.

For example, let $T = 1200$, $t_p = 30$, and $|A| = 3$. We have six input variables: one for each algorithm at time 10 seconds and one for each algorithm at $T_s = 1180$ seconds. We learn a Bayesian classifier that predicts the best algorithm at time 1180. The classifier is used on the learning set to define a hybrid algorithm that runs each pure algorithm for 10 seconds, uses the Bayesian classifier to predict which of the three will be best at 1180 seconds, and then runs the selected algorithm for the remaining 1170 seconds, continuing the search from where it left off. We repeat this procedure, learning a new Bayesian classifier for different values of $t_p \in \{60, 90, \dots, 1200\}$. This results in an assessment of hybrid algorithm performance for each prediction time. The prediction time with the best performance is t^* . The Bayesian classifier for $t_p = t^*$, is then used on the test set.

In all cases, ties among the pure algorithms are broken by selecting the algorithm with the best mean solution quality on the learning set at time T . The Bayesian classifiers are learned using WinMine 2.0 (Chickering 2002) with default parameters. We refer to this techniques as *Bayes*.

Reinforcement Learning for Switching

The switching paradigm allocates run-time to the pure algorithms during the search. In contrast to predictive selection, the allocation decision is made multiple times. N iterations are performed such that each iteration, i , has a time limit of t_i CPU seconds and $\sum_{1 \leq i \leq N} t_i = T$. During each iteration, the run-time of each pure algorithm, $a \in A$, is determined using a weight, $w_i(a)$, which is learned as the search progresses. The weights for an iteration are normalized to sum to 1 and therefore, $w_i(a)$ corresponds to the fraction of time t_i allocated to algorithm a . For example, if algorithm a has weight $w_i(a) = 0.2$ and $t_i = 60$ seconds, then a is run for 12 seconds during iteration i . Weights are initialized to $1/|A|$.

Weights are updated after each iteration by considering the current weight and the performance of each algorithm during the last iteration. Performance is measured in cost improvement per second, which allows us to compare algorithms despite the differing run times. We normalize the cost improvement per second to sum to 1 producing a performance value, $p_i(a)$. The weights are then adjusted using a standard reinforcement learning formula: $w_{i+1}(a) = \alpha \times w_i(a) + (1 - \alpha) \times p_i(a)$. The α value controls the

influence of the previous weight and the performance value.

Unlike the predictive paradigm, switching shares information among algorithms. Each invocation of an algorithm begins with the best solution found so far. However, different pure algorithms are able to exploit this information differently. Below, we define the exploitation for each pure algorithm.

Two switching variations are used here. In *rl-static*, the length of each iteration does not vary: $t_i = 60$, for all i . For *rl-double* the time for the first two iterations is fixed, $t_1, t_2 = 60$, and subsequently $t_{i+1} = 2 \times t_i$. We switch between algorithms $|A| - 1$ times during an iteration, running each algorithm exactly once. For each iteration, the order in which the algorithms are run is randomly generated with uniform probability. This was done because it is likely to be easier to improve a solution earlier in an iteration since later the easy improvements will already have been made by the earlier algorithms. Similarly, no weight updating is done until after the second iteration, as it is very easy to find large improvements in solution quality in the first iteration. In both *rl-static* and *rl-double*, $\alpha = 0.5$. Experimenting with different values of α and t_i is an interesting direction we intend to pursue in our future work.

Empirical Studies

Three sets of 20×20 job shop scheduling problems are used. A total of 100 problem instances in each set were generated and 60 problems per set were arbitrarily identified as the learning set for the Bayesian classifier. The rest were placed in the test set. The difference among the three problem sets is the way in which the activity durations are generated. In the **Rand** set durations are drawn randomly with uniform probability from the interval $[1, 99]$. The **MC** set has activity durations drawn randomly from a normal distribution. The mean and standard deviation are the same for the activities on the same machine but different on different machines. The durations are, therefore, machine-correlated (MC). In the **JC** set the durations are also drawn randomly from a normal distribution. The means and standard deviations are the same for activities in the same job but independent across jobs. Analogously to the MC set, these problems are job-correlated (JC). These different problem structures have been studied for flow-shop scheduling (Watson *et al.* 2002). 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.

Three pure algorithms are taken from the scheduling literature. These were chosen from a set of eight algorithms because they have generally comparable behavior on the learning set. The other techniques 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 (Nowicki & Smutnicki 1996).
- **texture**: a constructive search technique using texture-based heuristics (Beck & Fox 2000), strong constraint propagation (Nuijten 1994; Laborie 2003), and bounded

chronological backtracking. The bound on the backtracking follows the optimal, zero-knowledge pattern of 1, 1, 2, 1, 1, 2, 4, ... (Luby, Sinclair, & Zuckerman 1993). The texture-based heuristic identifies a resource and time point with maximum competition among 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 Set-Times heuristic (Scheduler 2001), the same propagation as texture, and slice-based search (Beck & Perron 2000), 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, ...

The best solution found so far is shared among the algorithms in the switching paradigm. The constructive algorithms only make use of the bound on the solution quality, searching for a solution that is strictly better than the best solution found so far. Each iteration of tabu-tsab, however, must begin with an initial solution. The best solution found so far is used.

Our control techniques assume that the pure algorithms are able to find a sequence of increasingly good solutions. As we are minimizing a cost function an initial solution (with a very high cost) is always easily available. Each algorithm successively finds better solutions as it progresses either through local search or branch-and-bound, terminating when it has either proved optimality or exceeded a time limit. At any given time, each algorithm can return the best complete solution that it has found so far. If, in a given time interval, no better solution has been found, the solution found in the previous time interval is returned. In the Bayes approach, for example, the binary variables are always constructed based on the best complete solution that the algorithm has found up to a time point.

Results

Table 1 displays the results on the three problem sets averaged over ten independent runs, R , with an overall time limit $T = 1200$. The evaluation criterion is 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:

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

where K is a set of problem instances, R is a set of independent runs with different random seeds, $c(a, k)$ is the lowest cost found by algorithm a on k , and $c^*(k)$ is the lowest cost known for k . Also displayed is the mean fraction of problems in each run for which each algorithm finds the best known solution.

	Rand		MC		JC		All	
	Frac. Best	MRE	Frac. Best	MRE	Frac. Best	MRE	Frac. Best	MRE
tabu-tsab	0.265	0.0211	0.385	0.0199	0.225	0.0157	0.292	0.0189
texture	0.228	0.0206	0.09	0.0292	0.703	0.0089	0.340	0.0196
settimes	0.0	0.0519	0.0	0.0460	0.325	0.0168	0.108	0.0382
Bayes	0.228	0.0186	0.230	0.0192	0.627	0.0074	0.361	0.151
rl-double	0.453	0.0141 [‡]	0.453	0.0142	0.825	0.0042 *	0.577	0.0108 [‡]
rl-static	0.093	0.0243 [†]	0.088	0.0265	0.754	0.0049	0.311	0.0185

Table 1: The performance of each technique on the test set. ‘*’ indicates significantly lower mean relative error (MRE) than all pure techniques, ‘‡’ indicates significantly lower MRE than all other algorithms, and ‘†’ indicates significantly worse performance than the best pure technique. “Frac. Best” is the mean fraction of problems in each set for which the algorithm found the best known solution.

Bayes is the only approach that uses the learning set. Recall that the learning set is used to build a Bayesian classifier for each possible prediction time and identify the one with the smallest MRE. The best t_p was found to be 270 seconds, meaning that each pure algorithm is run for 90 seconds and then the Bayesian classifier is used to select the pure algorithm to be run for the subsequent 930 seconds. The Bayes technique achieves an MRE that is not significantly¹ different from the best pure algorithm while finding the best solution in about the same number of problems as the best pure algorithm.

The two reinforcement learning algorithms also achieve strong performance. The better one, rl-double, performs significantly better than all other algorithms based on MRE and able to find the best known solution for over half the instances. One advantage of the switching paradigm is that it can use different pure algorithms in different parts of the search, and therefore, can achieve better performance on a problem instance than the best pure technique. In contrast, predictive selection techniques cannot perform better than the best pure technique. In fact, rl-double finds a better solution than any pure technique in 45 of the problems.

The rl-static technique performs significantly worse than rl-double. The doubling of the time in the iterations appears to be an important part of the performance of rl-double. Nonetheless, rl-static is able to achieve performance that is not significantly worse than the best pure technique. Furthermore, it finds a better solution than any pure technique in 16 of the problems.

The fact that we have created a technique that achieves significantly better performance than the best pure technique is not the main conclusion to be drawn from this experiment. Rather, the importance of these results stems from the fact that the technique does not depend on expertise either in terms of development of new, pure scheduling algorithms or in terms of development of a detailed domain and algorithm model. A low-knowledge approach is able to achieve better performance while *reducing* the required expertise.

Further Investigations

Our initial experiment demonstrates that, at least for the problem sets, algorithms, and time limit used, it is possi-

¹All statistical results are measured using a randomized paired- t test (Cohen 1995) and a significance level of $p \leq 0.005$.

ble to use low-knowledge approaches to algorithm control to achieve MRE performance that is significantly better than the best pure algorithm. A number of questions can now be asked with respect to these results. How sensitive are the results to different choices of parameters such as the overall time limit? Can we evaluate the extent to which the machine learning techniques are responsible for the performance of the control techniques?

Other Time Limits To address the issue of different time limits, we perform a series of experiments using the same design as above with $T \in \{120, 240, \dots, 1200\}$. Table 2 presents the results. The “Best Pure” column is the MRE of the best pure technique on the test set for each time limit. For $T \in \{120, \dots, 960\}$, texture achieves the lowest MRE of all pure techniques. For $T \geq 1080$, tabu-tsab is best. The MRE of Bayes is not significantly worse than the best pure algorithm except for $T = 120$. The rl-static approach achieves significantly lower MRE than all pure techniques for three time limits: $T \in \{240, 360, 480\}$. Finally, rl-double achieves MRE significantly lower than all pure techniques for $T \geq 360$ and comparable performance for $T \leq 240$. For $T \geq 600$, rl-double achieves significantly better performance than all other algorithms in Table 2.

We conclude that the results for $T = 1200$ are applicable to other choices for an overall time limit. The rl-double algorithm dominates regardless of time limit and the Bayes approach no worse than the best pure technique except for very low time limits.

Other Low Knowledge Techniques To evaluate the impact of the learning techniques in isolation, we create control variations that do not perform learning. We replace the Bayesian classifier with a simple rule: at prediction time, choose the algorithm that has found the lowest cost solution. The learning set is used to evaluate the performance of this *mincost* rule at each prediction time for each T value. The prediction time, t^* that results in the lowest MRE on the learning set for a given T is then used on the test set for that T . Similarly, to assess the reinforcement learning, we run rl-static and rl-double with $\alpha = 1$. Each pure algorithm receives an equal portion of the iteration time, regardless of previous performance. We call these “no-learning” variations, *nl-static* and *nl-double*. Finally, to investigate the impact of communication among the algorithms, the *no-com* technique is nl-double without sharing of the best known

Time Limit	Algorithm							
	Best Pure	Bayes	mincost	rl-double	rl-static	nl-double	nl-static	no-com
120	0.0387	0.0406 [†]	0.0409 [†]	0.0356	0.0340	0.0364	0.0347	0.0560 [†]
240	0.0311	0.0298	0.0315	0.0270	0.0246*	0.0278	0.0270	0.0440 [†]
360	0.0277	0.0258	0.0266	0.0216*	0.0211*	0.0232	0.0238	0.0387 [†]
480	0.0256	0.0232	0.0238	0.0191*	0.0197*	0.0203*	0.0225	0.0353 [†]
600	0.0241	0.0208	0.0216	0.0168[‡]	0.0190	0.0188*	0.0220	0.0330 [†]
720	0.0225	0.0196	0.0203	0.0147[‡]	0.0187	0.0174*	0.0218	0.0311 [†]
840	0.0215	0.0181	0.0193	0.0134[‡]	0.0186	0.0158*	0.0217	0.0298 [†]
960	0.0207	0.0169	0.0181	0.0125[‡]	0.0186	0.0146*	0.0217 [†]	0.0288 [†]
1080	0.0198	0.0158	0.0171	0.0116[‡]	0.0186	0.0139*	0.0217 [†]	0.0277 [†]
1200	0.0189	0.0151	0.0163	0.0108[‡]	0.0185	0.0132*	0.0217 [†]	0.0268 [†]

Table 2: The mean relative error (MRE) of each algorithm for different time limits. ‘*’ indicates an MRE significantly lower than all pure techniques at the same time limit, ‘[‡]’ signifies an MRE significantly lower than all other tested algorithms at the same time limit while ‘[†]’ indicates significantly worse than the best pure technique.

solution among the algorithms: each algorithm is allowed to continue from where it left off in the previous iteration.

The results of these approaches are also presented in Table 2. The mincost technique does not perform significantly worse than the best pure technique, demonstrating that even a simple rule can be applied in a low-knowledge context to achieve competitive performance. Bayes is significantly better than mincost only for $T = 960$ and $T = 1200$. So while the machine learning techniques do provide a benefit, it does not appear to be robust to different time limits.

Reinforcement learning has a stronger impact. Though not displayed in the table, rl-static is significantly better than nl-static for all T and rl-double is significantly better than nl-double at all time limits except $T = 120$. Interestingly, nl-double not only achieves significantly better performance than the pure techniques on seven of the ten time limits but also achieves a significantly lower MRE than all algorithms except rl-double for $T = 840$ and $T = 960$. Finally, no-com performs worse than the best pure algorithm at all time limits: communication between algorithms is clearly an important factor for performance.

Perfect High-Knowledge Classifiers We can perform an *a posteriori* analysis of our data to compare our low-knowledge approach with the best performance that can be expected from high-knowledge classifier which seek to select a pure algorithm for a given problem instance. The *best-subset* classifier chooses the pure algorithm that is best, on average, for the subset to which the test instance belongs. This is the best result we can achieve with a high-knowledge classifier that infallibly and with zero CPU time can categorize instances into the three problem sets. The *best-instance* makes a stronger assumption: that, with zero CPU time, the high-knowledge classifier can always choose the best pure algorithm for a given instance. This is the optimal performance that is achievable by *any* technique based on choosing the best pure algorithm for a given instance.

The rl-double technique achieves a lower MRE than best-subset for $T \geq 360$. Furthermore, rl-double performs as well as best-instance for all time limits $T \geq 360$. In other words, without complex model building our best low knowl-

Time Limit	Algorithm		
	rl-double	best-subset	best-instance
120	0.0356	0.0366	0.0313[‡]
240	0.0270	0.0291	0.0242[‡]
360	0.0216*	0.0259	0.0205
480	0.0191*	0.0234	0.0184
600	0.0168*	0.0219	0.0170
720	0.0147*	0.0203	0.0156
840	0.0134*	0.0190	0.0144
960	0.0125*	0.0180	0.0133
1080	0.0116*	0.0171	0.0122
1200	0.0108*	0.0164	0.0115

Table 3: The mean relative error (MRE) for different overall time limits. ‘*’ indicates a significantly lower MRE in the comparison of rl-double and best-subset while ‘[‡]’ signifies the same thing for the comparison of rl-double and best-instance.

edge approach achieves performance that is as good as the best possible high-knowledge classification approach.

Future Work

We have examined a number of algorithm control approaches on a single problem type. While there is no reason to expect JSPs to be so unlike other optimization problems that similar results would not be observed, it is necessary to evaluate low-knowledge approaches on other optimization problems. In addition, it may be more relevant to apply algorithm control to problems with particular characteristics. A real system is typically faced with a series of related problems: a scheduling problem gradually changes as new orders arrive and existing orders are completed. As the problem changes, algorithm control techniques may have the flexibility to appropriately change the manner in which underlying pure algorithms are applied.

We have examined a very simple version of reinforcement learning and the best control technique, rl-double, made no use of the off-line learning set. Two obvious uses for the off-line problems is as a basis for setting the α value and as a basis for iteration-specific weights. If it is true that differ-

ent pure algorithms are more appropriate for different phases of the search, we could use the learning set to weight each algorithm in each iteration. The issue of the ordering of the pure algorithms within an iteration is also an interesting area for future work. It is clear from the result of nl-double that the simple act of switching among a set of algorithms brings problem solving power. Understanding the reasons behind this may help to further leverage such simple techniques.

Finally, we intend to investigate the scaling of these techniques as the number of pure algorithms increases. It seems likely that the prediction-based techniques would suffer: prediction time will need to be longer and/or the time given to each pure technique will be reduced. These implications will both lead to poorer performance as less time is left for the chosen algorithm and the algorithm selection will be less accurate. In contrast, the reinforcement learning, especially if off-line learning is used to set iteration-specific weights, may be able isolate the subset of pure algorithms that are most useful.

Conclusion

We have shown that low-knowledge metrics of pure algorithm behavior can be used to form a system that consistently and significantly out-performs the best pure algorithm. Machine learning techniques play an important role in this performance, however, even a simpleminded approach that evenly distributes increasing run-time among the pure algorithms performs very well.

Our motivation for investigating low-knowledge algorithm control was to reduce the expertise necessary to exploit optimization technology. Therefore, the strong performance of our techniques should be evaluated not simply from the perspective of an increment in solution quality, but from the perspective that this increment has been achieved without additional expertise. We neither invented a new pure algorithm nor developed a detailed model of algorithm performance and problem instance features. Therefore, we believe low-knowledge approaches are an important area of study in their own right.

References

- Beck, J. C., and Fox, M. S. 2000. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence* 117(1):31–81.
- Beck, J. C., and Perron, L. 2000. 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)*.
- Chickering, D. M. 2002. The WinMine toolkit. Technical Report MSR-TR-2002-103, Microsoft, Redmond, WA.
- Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass.
- Horvitz, E.; Ruan, Y.; Gomes, C.; Kautz, H.; Selman, B.; and Chickering, M. 2001. A bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth Conference on uncertainty and Artificial Intelligence (UAI-2001)*, 235–244.
- Kautz, H.; Horvitz, E.; Ruan, Y.; Gomes, C.; and Selman, B. 2002. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, 674–681.
- Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.
- Lagoudakis, M. G., and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, 511–518. Morgan Kaufmann, San Francisco, CA.
- Le Pape, C.; Perron, L.; Régim, J.; and Shaw, P. 2002. Robust and parallel solving of a network design problem. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP02)*, 633–648.
- Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. 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)*, 556–572.
- Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47:173–180.
- Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *CONSTRAINTS* 1(1,2):7–43.
- Nowicki, E., and Smutnicki, C. 1996. A fast taboo search algorithm for the job shop problem. *Management Science* 42(6):797–813.
- Nuijten, W. P. M. 1994. *Time and resource constrained scheduling: a constraint satisfaction approach*. Ph.D. Dissertation, Department of Mathematics and Computing Science, Eindhoven University of Technology.
- Rice, J. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.
- Ruan, Y.; Horvitz, E.; and Kautz, H. 2002. 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)*, 573–586. Springer-Verlag.
- Scheduler. 2001. *ILOG Scheduler 5.2 User's Manual and Reference Manual*. ILOG, S.A.
- Watson, J.-P.; Barbulescu, L.; Whitley, L.; and Howe, A. 2002. Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. *INFORMS Journal on Computing* 14(1).
- Watson, J.-P. 2003. *Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem*. Ph.D. Dissertation, Dept. of Computer Science, Colorado State University.