Five Pitfalls of Empirical Scheduling Research

J. Christopher Beck*, Andrew J. Davenport‡, Mark S. Fox*‡
Department of Computer Science* and Department of Industrial Engineering‡
University of Toronto
Toronto, Ontario, CANADA
M5S 3G9

{chris, andrewd, msf}@ie.utoronto.ca

Abstract

A number of pitfalls of empirical scheduling research are illustrated using real experimental data. These pitfalls, in general, serve to slow the progress of scheduling research by obsfucating results, blurring comparisons among scheduling algorithms and algorithm components, and complicating validation of work in the literature. In particular, we look at difficulties brought about by viewing algorithms in a monolithic fashion, by concentrating on CPU time as the only evaluation criteria, by failing to prepare for gathering of a variety of search statistics at the time of experimental design, by concentrating on benchmarks to the exclusion of other sources of experimental problems, and, more broadly, by a preoccupation with optimization of makespan as the sole goal of scheduling algorithms.

Introduction

With the recent burgeoning of interest in empirical approaches to artificial intelligence [AAAI Empirical Workshop, 1994; ECAI Empirical Workshop, 1996], a number of authors have made calls for a more scientifically rigorous basis for such research [Hooker, 1994; Cohen, 1995]. While much of the field of the empirical study of algorithms is relatively immature, empirical scheduling research may be especially so due to the widespread (and currently unmet) demand for scheduling solutions to real problems in industry and elsewhere. This demand has significant positive impact for the research in terms of funding, sources of challenging problems, and opportunities to contribute beyond the academic world. However, the same demand can result in a retardation of the progress of the science through the strong temptation to concentrate on delivering an acceptable solution method for a particular problem rather than on developing an understanding of the relative merits of existing and novel techniques.

We do not claim that developing a novel solution method for a problem is, in itself, anti-productive for the science of scheduling. Indeed, much progress has been produced from such seminal work [Fox, 1983; Zweben et al., 1993]. However, having this mode of progress as the sole or primary motive force is characteristic of an immature field and leads to a balkanization of the research community: each research group has its own problem sets and solution techniques and, though these may both be published and available, there is little cross-fertilization.

A more progressive approach to scheduling research involves not only exploratory forays into new problem areas, but also rigorous adaptation and re-implementation of the work of other researchers, reproduction of results, and hypotheses testing to develop a deeper understanding of the behaviour of scheduling techniques across a wide body of problems. In short, we urge that the calls for a rigorous empirical science of algorithms be heeded particularly well in the scheduling community [Hooker, 1994].

In this paper we present a number of pitfalls that we have observed and experienced in our research. The data used to illustrate each pitfall comes from experiments we have performed over the past few years. The motivation for writing this paper is similar to that of earlier work describing the dangers of empirical testing in the design of algorithms for SAT [Gent et al., 1997]: we hope that by describing our experiences we will further the progress and maturity of the research.

Pitfall I: The Scheduling Monolith

Consider the following experiment, in which two scheduling algorithms, Monolith1 and Monolith2, are compared on a set of problems from the Operations Research library [Beasley, 1990]. Five different CPU time bounds (10 minutes to 50 minutes) are used and the mean relative error[1] from optimal for which each algorithm was able to find a solution is displayed in Figure 1. (See [Beck et al., 1997b] for a full description of this experiment using the 20 minute CPU time bound.) The results are clear-cut: Monolith1 significantly outperforms Monolith2 at each time limit.

Based on this experiment, we can make conclusions concerning the relative abilities of the two algorithms: Monolith1 is better than Monolith2 (at least on the problems tested—see Pitfall IV). There are, however, two major difficulties with the experiment:

1. We are unable to achieve any deeper understanding of the scheduling algorithms because conclusions beyond those comparing the algorithms as a whole are not justified. In particular, this experiment provides no insight as to *why* Monolith1 outperforms Monolith2.
2. If we know the components of Monolith1 (*e.g.,* the heuristic or backtracker that is used), as is common, it is tempting to draw the conclusion that a particular part of the Monolith1 algorithm is responsible for the performance differences.

Algorithms Monolith1 and Monolith2 are actually both of the form shown in Figure 2. Both algorithms use constraint propagators: temporal arc-B-consistency [Lhomme, 1993], constraint-based analysis (CBA) [Erschler et al., 1976; Erschler et al., 1980], and edge-finding [Carlier and Pinson, 1989; Nuijten, 1994]. The differences between the two algorithms are how the heuristic decisions are made (line 7) and how a commitment (search decision) is retracted at a dead-end (line 9).

- Monolith1: uses a heuristic based on the SumHeight contention estimation algorithm [Beck et al., 1997b] and limited discrepancy search (LDS) [Harvey, 1995; Harvey and Ginsberg, 1995] for retraction.

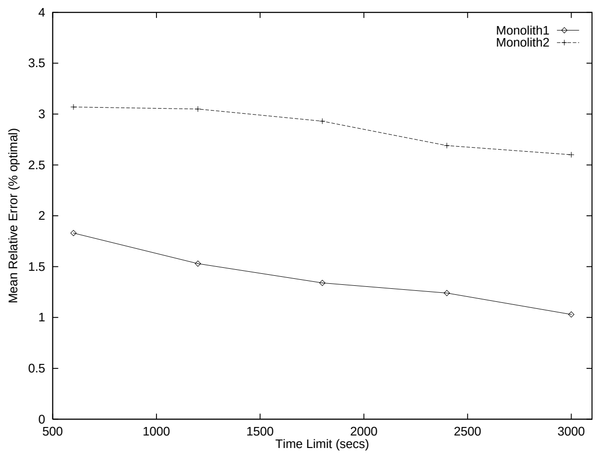

Figure 1. Results from the Monolith Experiment

1. Mean relative error (MRE) is the mean smallest amount above the optimal makespan that an algorithm could find a solution expressed as a percentage of the optimal makespan.

- Monolith2: uses the CBASlack heuristic [Smith and Cheng, 1993; Cheng and Smith, 1996] and chronological backtracking.

With this internal information, we see that, due to poor design, the experiment does not provide any insight as to why Monolith1 is better than Monolith2: is it the heuristic or is it the retraction method?

An obvious solution, if we are to examine which heuristic is better, is to use a more rigorous, non-monolithic experimental design. More revealing results are displayed in Figure 3. The results from Figure 1 (Monolith1 is now identified by SumHeight+LDS, Monolith2 is now CBASlack+Chron) are re-displayed together with two other algorithms: CBASlack with LDS (CBASlack+LDS) and SumHeight with chronological backtracking (SumHeight+Chron). These results show an interesting interaction between the heuristic and retraction techniques. With chronological backtracking SumHeight performs significantly better than CBASlack, however this difference disappears when using LDS.

Our point is not the actual comparison of SumHeight and CBASlack (see [Beck et al., 1997b] for a comparison) but rather that:

```
finished := false
while(finished = false){
   edge-finding
   if (edge-finding makes no commitments)
      CBA
   if (no commitments from CBA or from edge-finding)
      make heuristic commitment
   if (dead-end)
      retract some commitment
   else
      arc-B-consistency temporal propagation
   if (all-activities-sequenced OR CPU limit reached)
      finished := true
}
```

Figure 2. The Template for Monolith1 and Monolith2

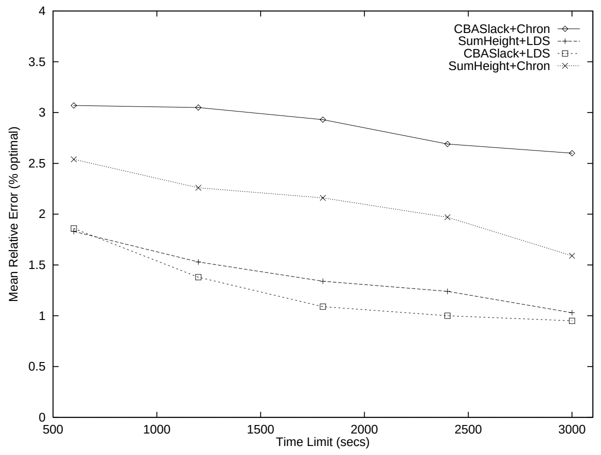

Figure 3. More Revealing Results from a Non-monolithic Experiment

1. Any claim of superiority of the heuristics based on the results of the monolithic experiment is unjustified.
2. The monolithic experiment, even when the components of the algorithms are known, may hide interesting data such as the interactions between heuristics and retraction techniques that we see in Figure 3.

Such experiments do appear in the literature due, partially, to the fact that researchers often work on whole algorithms and test new algorithms against existing ones. For example, [Smith and Cheng, 1993] compare four algorithms:

• PCP – using temporal arc-B-consistency and CBA propagation techniques, the CBASlack heuristic which posts precedence constraints between activities, and no backtracking.

• ORR/FSS [Sadeh, 1991] – using the ORR/FSS heuristic which assigns start-times to activities, temporal arc-B-consistency and resource arc consistency propagation, and chronological backtracking.

• ORR/FSS+ [Xiong et al., 1992] – the same as ORR/FSS using a form of intelligent backtracking instead of chronological backtracking.

• CPS [Muscettola, 1992] – using a heuristic which posts unary temporal constraints, temporal arc-B-consistency, resource arc consistency, and restart backtracking.

Experimental results showed that PCP was competitive (in terms of number of problems solved) with the other algorithms while using significantly less CPU time. In the comparison of scheduling *algorithms*, we agree with these results and the interpretation. However, among the four algorithms there are four different retraction techniques (chronological backtracking, a form of intelligent backtracking, restart, and no backtracking), three different sets of propagation techniques, and four different types of heuristic commitments (*e.g.*, assigning start-times, posting precedence constraints). On that basis, we do not believe the following conclusion is justified: "Evaluation … has shown that our heuristics provide comparable results at very low computational expense." [Smith and Cheng, 1993, p. 144]. The results may be due to the CBASlack heuristic as claimed, or to the CBA propagation or to the differing types of commitments that each heuristic makes. What happens if we use the CBA propagator with the ORR/FSS heuristic? What happens if we post precedence constraints with CPS?

As a second example, [Nuijten, 1994] compares the SOLVE algorithm (using a randomized heuristic commitment technique to assign start-times, sophisticated propagation (edge-finding, temporal and resource arc consistency), and bounded chronological backtracking with restart) against ORR/FSS (as described above) and against ORR/FSS augmented with the propagation techniques used by SOLVE. The results showed that SOLVE strongly outperforms augmented ORR/FSS which in turn strongly outperforms ORR/FSS. From the comparison of augmented ORR/FSS and ORR/FSS, it is observed that the sophisticated propagation techniques contribute significantly to the scheduling algorithms. In comparing SOLVE and augmented ORR/FSS, however, there are two candidates for the performance difference: the heuristic commitment technique and the retraction technique. Nuijten states that these results do not mean that sophisticated heuristics such as those used in ORR/FSS are not useful, however, despite this statement, the results cast them in a poor light given that they were not directly tested.

In our analysis of existing constraint-directed scheduling techniques [Beck, 1997], we have identified, four key components present in many scheduling algorithms:

• **Commitment Type** We take the general view that a commitment is a set of constraints added to the constraint graph. However, we expect that different types of constraints (*e.g.*, assigning start-times by adding unary equals constraints, sequencing activities by adding binary precedence constraints) will have different effects on scheduling performance. Given that the same heuristic techniques can be used to make different types of commitments, the commitment-type is an important and orthogonal component of a scheduling algorithm.

- **Propagators** A propagator is an algorithm that analyzes the current search state to find new constraints that are logically implied by, but not explicitly present in, the constraint graph. Examples of propagators include the arc-B-consistency, CBA, and edge-finding noted above.
- **Heuristic Commitment Techniques** Heuristic commitment techniques are algorithms that heuristically suggest that a constraint or a set of constraints, though not necessarily implied by the current search state, are likely to lead to a solution. Above we noted the CBASlack heuristic and the heuristic based on the SumHeight contention estimation algorithm.
- **Commitment Retraction Techniques** When a dead-end (*i.e.,* a search state where one or more existing constraints is broken) is reached, the commitment retraction technique identifies a previously made commitment or commitments to be removed from the constraint graph and determines how to handle intervening commitments (*i.e.,* those made between the dead-end state and the state chosen to backtrack to). For example, chronological backtracking and LDS are two retraction techniques.

It is critical that researchers isolate the components of a scheduling algorithm and perform experiments that compare components of the same type. Furthermore, such comparison can not be done simply under one experimental design (*i.e.,* by holding all the other components constant) because, as demonstrated in Figure 3, there is the possibility of interactions among components. Identifying and investigating these interactions is an invaluable addition to understanding at a deep level the behaviour of search on a scheduling problem.

Pitfall II: Measuring CPU Time

Unfortunately, even when taking into account different components of a scheduling algorithm, the basis upon which competing algorithms are evaluated may be a source of errors, confusion, and difficulty. Consider the following experiment designed to test the effectiveness of two heuristic commitment techniques: CBASlack and SumHeight.

The two heuristic commitment techniques are tested with the propagators noted above (CBA, edge-finding, temporal arc-B-consistency) and with the LDS retraction component. The problems are 5 sets of 60 job shop scheduling problems, with sizes of {10×10, 12×12, 15×15, 18×18, 20×20} generated using Taillard's problem generator (Taillard, 1993). A CPU time bound of 20 minutes was allowed for each algorithm on each problem. If the bound was reached, failure on that problem was returned. Results in terms of number of problems solved are shown in Figure 4. The mean CPU time for the problems that both algorithms solved is displayed in Figure 5.

These results indicate that CBASlack and SumHeight perform about equally on these problem sets. However, Figure 6 and Figure 7 show a different perspective on the same experiment. On problems solved by both algorithms, we plot the mean number of implied commitments (those made by a propagator) in Figure 6 and the mean number of heuristic commitments in Figure 7.

Contrary to appearances the only significant difference between the implied commitments is for problems of size 10×10. Figure 6 and Figure 7 demonstrate that by far most of the commitments are implied commitments. Given the shape similarities between Figure 6 and Figure 5, we also might infer that the CPU time is dominated by the propagators.[2] Using CPU time, therefore, when our goal is to compare the heuris-

2. Figure 5 and Figure 6 do not directly amount to evidence of the dominance of CPU time by the propagators, however, two additional pieces of information help in the inference. First, we have implemented Nuijten's edge-finding algorithm [Nuijten, 1994] which has a time complexity, in each search state, of $O(mn^2)$ (m is the number of resources, n the number of activities per resource) in both average- and worst-case. This is greater than the worst-case complexity of SumHeight and the average-case complexity of CBASlack. Second, anecdotal evidence indicates that, in our implementation, over 30% of the CPU time is spent in edge-finding alone.

tics does not appear to be helpful. Differences in the heuristic performance are not reflected in the overall CPU time. In particular, CPU time hides the fact that, as shown in Figure 7, SumHeight makes significantly fewer heuristic commitments on each problem set and that SumHeight makes significantly more implied commitments per heuristic commitment on each problem set, as can been seen by comparing Figure 6 and Figure 7.

One view is that if our goal is to solve as many problems as we can in as short a time as we can, the number of heuristic commitments made is not a relevant statistic. We are interested in the bottom line performance and whether that performance is achieved with implied, heuristic, or divine commitments is not important. This view, typical of end-users of scheduling systems, is important especially if we want the systems we are developing to have industrial relevance. However, given that we are developing systems and that experiments must be on some set of problems, differences such as the number of heuristic commitments or the ratio of implied commitments to heuristic commitments are important. We take, for instance, the commitment ratio results to indicate that SumHeight is able to better identify critical, highly constrained areas in the constraint

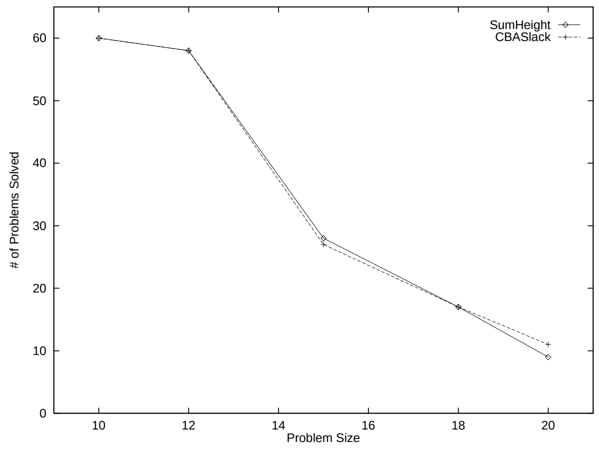

Figure 4. Number of Problems Solved vs. Problem Size

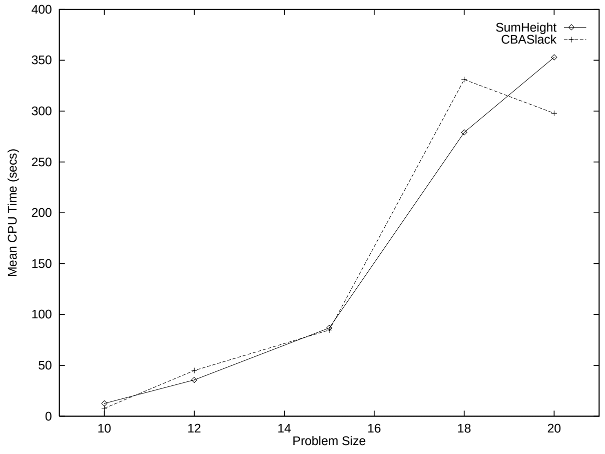

Figure 5. Mean CPU Time for Problems Solved by Both Algorithms

graph than CBASlack and further argue that this ability is important and will be reflected in bottom line performance as we move toward larger, industrial problems [Beck et al., 1997a].

The basis upon which algorithms are compared is an important issue in empirical scheduling research and, indeed, empirical algorithm research more generally. Comparisons based on overall CPU time, though important, lead to a number of issues [Hooker, 1996] such as the need to compare research code against industrial code and therefore to spend time in the software engineering of code optimization rather than on research. As we have shown, an additional difficulty is that overall CPU time may not speak to a comparison of the components of a scheduling algorithm in which the researcher is interested.

Conversely, CPU time can be very relevant in decisions as to the viability of scheduling approaches. For example, we have observed a three order of magnitude improvement in CPU time in algorithms when we have restructured them to optimize average time complexity. This has had significant impact in the research that we have pursued. Results that indicate an average performance of, say 200 CPU seconds, on small job

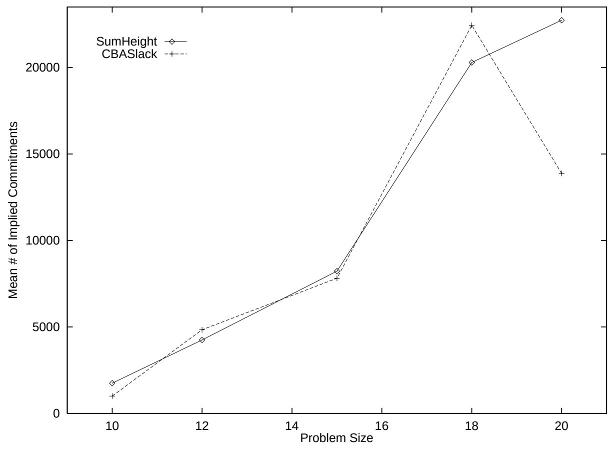

Figure 6. Mean Implied Commitments for Problems Solved by Both Algorithms

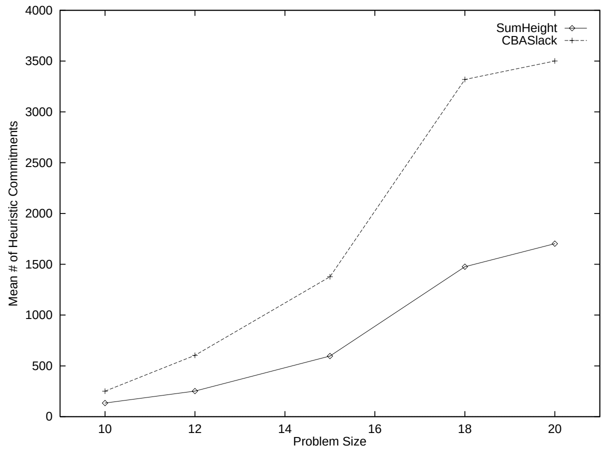

Figure 7. Mean Heuristic Commitments for Problems Solved by Both Algorithms

shop problems, might be taken as evidence that our approach is not going to be able to address larger industrial problems and so we may choose to investigate other options. In contrast, if that performance is 0.2 CPU seconds, our confidence in our methods and their scalability is significantly higher.[3]

We find that both of these arguments have merit and propose a balanced approach. Comparisons should be done both using CPU time and other search statistics (*e.g.*, number of commitments, number of backtracks, the ratio of commitments to backtracks). In the end, gathering and analyzing all these statistics will significantly aid in diagnosing performance and developing a deeper understanding of the algorithm behaviour. Even if CPU time is the main (or only) industrially relevant statistic, in the long term a balanced basis for evaluation will lead to a better understanding and, as a result of the deeper understanding, hopefully, to better CPU performance.

Pitfall III: Measuring Other Search Statistics

One of the problems with gathering multiple search statistics is that the experimental design can effect the validity of the data gathered. In optimization problems, it is common to compare algorithms on the basis of the mean relative error from optimal. Satisfaction algorithms can be tested on optimization problems by using a time bound and attempting to solve the problem at its known optimal. If the algorithm fails, the problem is relaxed (by some percentage of the optimal) and the algorithm is run again. This process is repeated until the problem is solved. The researcher then reports the mean relative error: the mean percentage above the optimal cost at which an algorithm was able to find a solution.

Unfortunately, this method can lead to mistaken assumptions about search statistics other than mean relative error. The difficulty arises because each algorithm (potentially) solves a completely different set of problems. On a single problem in the set, one algorithm may solve it to optimality while another may solve it to 1% of optimality, and a third only to 5%. Each algorithm has solved a different problem. If the problem characteristics change at different percentages of the optimal, this experimental design may have significant impact on results and conclusions. In particular, the interpretation of potentially interesting search statistics other than mean relative error may be suspect.

Table 1 presents a set of results from such a mean relative error experiment designed to test the efficacy of a number of heuristic commitment techniques.[4] A number of potentially interesting search statistics were gathered with the intention of using them subsequently to diagnose search behaviour. Because we are running propagators as well as a heuristic commitment component, an obvious statistic is to identify how many of the commitments are implied, that is, found by propagators. We also calculate the number of commitments for each probe in the search tree (Commitments/Probe), the number of heuristic commitments for each probe (HC/Probe), and the number of implied commitments for each heuristic commitment (IC/HC). A probe is defined to be the sequence of consecutive commitments not interrupted by a dead end.

In examining these statistics we may be able to develop insights into performance differences. Why, for example, was SumHeight able to solve problems significantly closer to optimal than LJRand? We observe that SumHeight has significantly fewer commitments per probe and heuristic commitments per probe. A possible explanation for the performance difference is that the SumHeight heuristic is able to detect dead-

3. The argument here is not that because we have good performance on small problems our algorithms will necessarily scale to industrial problems. Rather it is that if we have poor performance on small problems, there is little hope that the algorithms will improve when applied to larger problems. Another pitfall, however, that we do not have space to discuss here, is the common assumption that algorithms that show good results on small problems will scale up. Our experience with large industrial problems suggests that this is not necessarily the case.

4. See [Beck et al., 1997b] for a discussion of each of these heuristic commitment techniques.

ends earlier and therefore backtrack earlier, wasting less time in a sub-tree that will eventually prove fruitless. Unfortunately, an alternative explanation is that SumHeight solved problems with a smaller makespan (*i.e.*, a smaller mean relative error) which correlates highly with how constrained the problem is. Because the problems are more constrained, SumHeight is able to detect dead-ends sooner than LJRand on less constrained problems. The results may not have anything to do with a difference between SumHeight and LJRand but rather between the problems each solved.

To take another example, in Table 1 we see that SumHeight averages almost three times the number of implied commitments per heuristic commitment than CBASlack. It is tempting to conclude that the reason that SumHeight is able to find better solutions is that it is better able to identify and make commitments in highly constrained areas of the graph. This argument is again hampered by the experimental design. CBASlack solved problems with a larger makespan, which therefore are less constrained than those solved by SumHeight. Based on the results in Table 1 it is not justified to attribute the success of SumHeight to the ability to identify highly constrained sub-graphs. The problems solved by SumHeight are more highly constrained and so the IC/HC ratio may simply be an artifact of the problems that were solved.

There do not appear to be easy ways to avoid having experimental design affect the validity of the search statistics. This is especially the case if the gathering of a particular statistic was not planned at design time. While design-for-measurement is important and preferable, unfortunately, it is not always the case that the researchers foresee all the data that might be interesting before the experiment is designed and run. A pragmatic approach suggests care with experimental design to take into account statistics that might be of interest and the less satisfying practice of *a posteriori* vigilance: researchers need to be particularly aware of alternative explanations arising from artifacts of experimental design.

Heuristic	Relative Error	Commitments (HC, IC)[a]	Backtracks	Commitments/ Probe	HC/ Probe	IC/ HC
Sum-Height	2.26	29751 (1426, 28325)	676	87.13	8.93	20.08
CBA-Slack	3.05	10215 (1162,9053)	400	120.74	35.45	6.99
First-Commit	4.49	25650 (1494, 24155)	947	112.79	5.78	19.15
LJRand	10.54	1329 (144, 1184)	9	448.64	50.32	7.86

Table 1: Mean Search Statistics for MRE Experiment with CPU Time Limit of 1200 Seconds

a. HC: Heuristic Commitment, IC: Implied Commitment

Pitfall IV: Dangers of Benchmarking

Until recently, at least within the AI community, there were no widely used scheduling benchmarks. With Sadeh's problem set [Sadeh, 1991] and the adoption of the OR-library benchmark set [Beasley, 1990] this has changed.[5] This is not to say that problem sets were not previously available, but rather than there was little or no effort to cross-

5. ORLIB has existed for sometime within the OR community, but it is only in the past few years, due primarily to cross-fertilization from the OR world, that it has been used in the AI community.

validate scheduling strategies on problems generated by other researchers. We view the rise of benchmark sets that are used by a variety of researchers as a positive step in the maturity of the field; however this step is not without its perils. Chief among the perils is a concentration solely on existing benchmarks rather than their use as part of the overall evaluation strategy.

To illustrate a simple interpretation of special case results, we use data from an (unpublished) pilot experiment. The experiment was designed to test the efficacy of a new idea for retraction of commitments that had been developed in our lab, TestBT.[6] We decided to test the new retraction technique against three existing techniques: LDS, chronological backtracking, and random backtracking (RandomBT—randomly select a previously made heuristic commitment, backtrack to it (undoing all intervening commitments) and make an alternative commitment). Each algorithm used the same heuristic commitment technique and propagators.

We had at our disposal the two benchmark sets noted above, Sadeh's problems and the ORLIB set. We first used Sadeh's problems and found that 53 of the 60 problems could be solved with no backtracking. This cut our problem set down to the remaining 7 problems. Table 2 shows the results of running three of the backtracking techniques on the problems (chronological backtracking was not able to solve any of these problems within a bound of 1000 backtracks).

Problem	TestBT		RandomBT		LDS	
	Commitments	BTs	Commitments	BTs	Commitments	BTs
1	291	11	191	3	172	1
2	427	20	203	4	197	2
3	748	48	251	6	203	2
4	288	12	157	1	205	2
5	745	24	179	1	642	7
6	1086	26	1386	42	2593	69
7	237	12	170	2	155	1

Table 2: Number of Commitments and Number of Backtracks (BTs) for 3 Retraction Techniques

Based on these results, TestBT does not look promising—it is even worse than random! Fortunately, we also tested LDS which we had reason to believe would perform better than RandomBT [Harvey, 1995]. The fact that RandomBT is competitive with LDS leads us to believe that the problem set is distorting our results. Possible explanations for these results include the small sample size and/or a floor effect: the problems are just too easy to show any differences among the retraction techniques.

To further investigate the efficacy of TestBT, we used 13 of the ORLIB problems. Using a 600 CPU second time bound, each algorithm attempted to solve each problem at its optimal makespan. If it was unsuccessful, the makespan was expanded by 0.005 times optimal. This continued until 1.1 times optimal, where, if the algorithm could not

6. We do not specify the actual retraction technique as it is irrelevant to this discussion and would detract from our main point.

solve the problem, it gave up. Figure 8 shows the results of this experiment (relative error values of 1.11 are used to indicate that the algorithm did not solve the problem at all within the time and makespan bounds).

Results in Figure 8 are very different from Table 2. Now the best algorithm (in terms of finding the closest to optimal makespan) is always either LDS or TestBT while the worst algorithm is always either chronological backtracking or RandomBT. On the basis of these results, we are lead to believe that exploration of TestBT is justified.

Generally, we have one or more of the following goals when running experiments:

- The ability to draw conclusions that are applicable to problems outside the benchmark set. Our benchmark sets, therefore, need to be representative of some interesting population of scheduling problems.
- The ability to draw conclusions that are applicable (or extendible) to industrial scheduling problems. Industrial problems may be a particular case of an interesting population; however we note them explicitly due to the scope of characteristics that have not been rigorously addressed (see Pitfall V).
- The ability to compare various algorithms.

It is unlikely that one benchmark set can be used toward all these goals. Furthermore, the relative evaluation of algorithms may change depending on the priority of these aims. With these goals in mind, we suggest the following types of problems should make their way into future benchmark sets.

1. **Randomly generated problems** An algorithm independent notion of difficulty (such as the identification of a phase transition [Cheeseman et al., 1991; Gent et al., 1996; Beck and Jackson, 1997]) requires statements about the entire space of problems. Therefore, sets of randomly generated problems of varying sizes are a necessary vehicle of demonstration.
2. **Structured randomly generated problems** Structured problems that are generated with some random component are also necessary. The structure is likely to stem both from the empirical notions of difficulty (*e.g.*, the intuition that bottleneck resources lead to more difficult problems [Sadeh, 1991]), and from structures in and characteristics of real world problems.

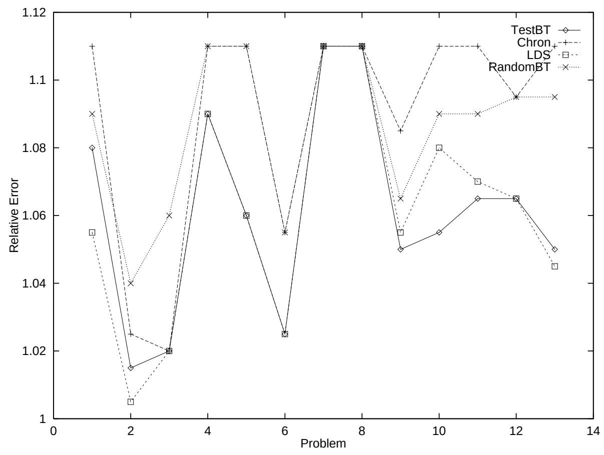

Figure 8. Relative Error on 13 ORLIB Problems

3. **Standard benchmarks** As noted above, standard benchmark problems are becoming widely used in the scheduling community. While these sets are unlikely to be representative of the entire space of problems, their usefulness as data points, given the mass of work that has addressed the problems, is significant.

Pitfall V: Makespan Considered Harmful

A lot of research and CPU cycles have been devoted to the optimization of makespan in job shop problems while, with a few notable exceptions [Saks, 1992; Nuijten, 1994; Le Pape, 1994b; Brucker and Thiele, 1996; Caseau and Laburthe, 1996; Nuijten and Aarts, 1997], little has been done on expanding the scope of constraint-directed scheduling technology much beyond the job shop. This is particularly distressing since, from the inception of constraint-directed approaches to scheduling, it has been recognized that real industrial scheduling is not simply meeting due dates, but rather satisfying many complex (and interacting) constraints from disparate sources within the plant and, indeed, the enterprise as a whole [Fox, 1983; Fox, 1990]. In fact, this was one of the original reasons to believe that scheduling was a prime application of constraint technologies.

It is unclear whether our preoccupation with makespan has allowed us to make many in-roads into the realities of scheduling problems that have existed for decades in industrial settings. Consider the following examples:

- **Varying release dates and due dates** Jobs are released for execution and due to be completed at varying time points over the scheduling horizon. While many advances in makespan optimization (*e.g.,* edge-finding) carry over to these problems, it is not clear if they will have as much of an impact on this class of problems. Furthermore there has been little work of which we are aware that examines techniques (*e.g.,* problem decomposition) that might be especially applicable to such problems. The addition of independent release times and due dates for different jobs is a trivial addition to the job shop model, yet it is unclear what techniques for makespan optimization can be carried over.
- **Advanced temporal constraints** The overwhelmingly most popular temporal relation among activities in scheduling research is the precedence constraint. Given that the 13 Allen relations [Allen, 1983] (*e.g.,* meets, during) have been recognized for many years and have been noted in many scheduling papers, it is surprising how little they have been addressed. Industrially such constraints arise in contexts such as curing, cooling, and spoilage, among others. These constraints have been so ignored that simple-minded techniques can result in surprising gains in scheduling ability in problems that contain such constraints [Davenport et al., 1997].
- **Inventory** Inventory is produced and consumed (often at varying rates) by activities in a manufacturing setting. Inventory must be stored and there may be a host of constraints, both temporal and capacity-based, with respect to how long and where it can be stored. While existing systems (notably ILOG Schedule [Le Pape, 1994b; Le Pape, 1994a] and KBLPS [Saks, 1992]) represent inventory, the only published results appear to be algorithms that are crafted for specific problems rather than generally addressing scheduling with inventory.
- **Time-varying constraints** Constraints, especially resource and inventory constraints in the manufacturing context, change over the scheduling horizon. The simplest example of this is scheduled down-time for a resource. More complicated examples exist, including time varying supply and demand (*e.g.,* in a seasonal business it is desirable to enforce a minimum inventory level that increases as the predicted high-season approaches), changes in staffing (*e.g.,* “learning” time during which new operators are not able to perform at peak speed or the fact that more people call in sick on Friday), and shift-specific operations (*e.g.,* clean-outs can only be done at night or on the weekend).

- **Production choices** There may be many ways to produce any particular finished good or intermediate product. The activity model from raw materials to final product is not, in fact, a simple acyclic temporal network (as it is often modeled) but an acyclic graph with specific choice points. These choices may represent trade-offs in quality, cost of production, or time, and are subject to resource and inventory use. Until we have a partial, evolving schedule, we do not know which choices should be made at these points.

These examples are low-level issues concerning specific scheduling constraints that have received little treatment in the literature. There are also a host of higher-level issues such as the robustness of schedules and the ease of rescheduling at execution time [Ow et al., 1988; Drummond et al., 1994; Hildum, 1994], relaxation of constraints in over-constrained problems [Beck, 1994], and, more generally, what constitutes a good solution. For example, imagine a system that creates schedules with a probabilistic measure of the ability to execute the schedule. Rather than guaranteeing a executable schedule (as is part of the *definition* of a solution in the research world), the system guarantees that the schedule can be executed with a probability of, say, 95%. In the current view of scheduling research, such a scheduler would not solve any problems at all, yet related concepts (*e.g.*, probabilistic customer service levels[7]) are commonly used in the industrial setting.

Constraints can model scheduling problems in a much more flexible, accurate, and representational way than other methods, for example, that rely on mathematical programming techniques. Why then are we spending an inordinate effort in attempting to squeeze the last 0.5% from the makespan of a particular instance of a scheduling problem?

Conclusion

Our purpose in this paper is to raise points for discussion with the eventual goal of facilitating the maturation of the field of empirical scheduling research. We believe that wide awareness of these pitfalls will encourage such maturation and make advances easier to achieve and validate.

In this paper, we have looked at five pitfalls that we have observed in our work in empirical scheduling. In particular, using data from actual experiments, we examined:

- the view of a scheduling algorithm as a monolithic whole,
- the use of CPU time as a performance measure,
- how experimental design can effect search statistics,
- the use of benchmarks, and finally,
- the use of makespan optimization as a primary goal in scheduling research.

The goal of scheduling research is to increase the ability to solve interesting problems. The source of the interest (*e.g.*, industrial need) may be important; however greater and more regular progress will be made if we can form an understanding of the behaviour of scheduling techniques on different problem types. This understanding will arise neither out of a concentration on delivering a solution to a particular type of problem nor out of a wholly abstract approach. A balanced approach suggests two research engines: application of theory to real world problems and generalization of practical advances to theoretical concepts. We believe that an understanding of pitfalls such as we have described here will aid in both of these efforts.

7. A probabilistic customer service level of 95% allocates inventory to a warehouse such that all customer orders will be met at least 95% of the time.

Acknowledgments

This research was funded in part by the Natural Science and Engineering Research Council of Canada, Numetrix Limited, the IRIS Research Network, the Manufacturing Research Corporation of Ontario, and Digital Equipment of Canada.

Thanks to Ed Sitarski, Angela Glover, and the anonymous reviewers for comments on and discussion of earlier drafts of this paper.

References

AAAI Empirical Workshop (1994). *Proceedings of the AAAI-94 Workshop on Experimental Evaluation of Reasoning and Search Methods*.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Beasley, J. E. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072. Also available by ftp from ftp://graph.ms.ic.ac.uk/pub/paper.txt.

Beck, J. C. (1994). A schema for constraint relaxation with instantiations for partial constraint satisfaction and schedule optimization. Master's thesis, Department of Computer Science, University of Toronto.

Beck, J. C. (1997). A generic framework for constraint-directed search and scheduling. Technical report, Department of Industrial Engineering, University of Toronto, 4 Taddle Creek Road, Toronto, Ontario M5S 3G9, Canada.

Beck, J. C., Davenport, A. J., Sitarski, E. M., and Fox, M. S. (1997a). Beyond contention: extending texture-based scheduling heuristics. In *Proceedings of AAAI-97*. AAAI Press, Menlo Park, California.

Beck, J. C., Davenport, A. J., Sitarski, E. M., and Fox, M. S. (1997b). Texture-based heuristics for scheduling revisited. In *Proceedings of AAAI-97*. AAAI Press, Menlo Park, California.

Beck, J. C. and Jackson, K. (1997). Constrainedness and the phase transition in job shop scheduling. Technical report, School of Computing Science, Simon Fraser University.

Brucker, P. and Thiele, O. (1996). A branch & bound method for the general-shop problems with sequence dependent set-up times. *OR Spektrum*, 18:145–161.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Caseau, Y. and Laburthe, F. (1996). Cumulative scheduling with task intervals. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*. MIT Press.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the really hard problems are. In *Proceedings of IJCAI-91*, volume 1, pages 331–337.

Cheng, C. C. and Smith, S. F. (1996). Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 1. Forthcoming.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass.

Davenport, A. J., Beck, J. C., and Fox, M. S. (1997). Propagation over the meets temporal constraint. Technical report, Department of Industrial Engineering, University of Toronto.

Drummond, M., Bresina, J., and Swanson, K. (1994). Just-in-case scheduling. In *Proceedings of AAAI-94*, pages 1098–1104, Menlo Park, CA. AAAI Press/MIT Press.

ECAI Empirical Workshop (1996). *Proceedings of the ECAI-96 Workshop on Empirical Artificial Intelligence*.

Erschler, J., Roubellat, F., and Vernhes, J. P. (1976). Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research*, 24:772–782.

Erschler, J., Roubellat, F., and Vernhes, J. P. (1980). Characterising the set of feasible sequences for n jobs to be carried out on a single machine. *European Journal of Operational Research*, 4:189–194.

Fox, M. S. (1983). *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA. CMU-RI-TR-85-7.

Fox, M. S. (1990). Constraint-guided scheduling - a short history of research at CMU. *Computers in Industry*, 14:79–88.

Gent, I. P., Grant, S. A., MacIntyre, E., Prosser, P., Shaw, P., Smith, B. M., and Walsh, T. (1997). How not to do it. Technical Report 97.27, School of Computer Studies, University of Leeds.

Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1996). The constrainedness of search. In *Proceedings of AAAI-96*, volume 1, pages 246–252.

Harvey, W. D. (1995). *Nonsystematic backtracking search*. PhD thesis, Department of Computer Science, Stanford University.

Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of IJCAI-95*, pages 607–613.

Hildum, D. W. (1994). *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA. 01003-4610. UMass CMPSCI TR 94-77.

Hooker, J. N. (1994). Needed: An empirical science of algorithms. *Operations Research*, 42:201–212.

Hooker, J. N. (1996). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42.

Le Pape, C. (1994a). Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66.

Le Pape, C. (1994b). Using a constraint-based scheduling library to solve a specific scheduling problem. In *Proceedings of the AAAI-SIGMAN Workshop on Artificial Intelligence Approaches to Modelling and Scheduling Manufacturing Processes*.

Lhomme, O. (1993). Consistency techniques for numeric CSPs. In *Proceedings of IJCAI-93*, volume 1, pages 232–238.

Muscettola, N. (1992). Scheduling by iterative partition of bottleneck conflicts. Technical Report CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University.

Nuijten, W. and Aarts, E. (1997). A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*. To appear.

Nuijten, W. P. M. (1994). *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Ow, P. S., Smith, S. F., and Thiriez, A. (1988). Reactive plan revision. In *Proceedings of AAAI-88*, pages 77–82. AAAI.

Sadeh, N. (1991). *Lookahead techniques for micro-opportunistic job-shop scheduling*. PhD thesis, Carnegie-Mellon University. CMU-CS-91-102.

Saks, V. (1992). Distribution planner overview. Technical report, Carnegie Group, Inc., Pittsburgh, PA, 1522.

Smith, S. F. and Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings AAAI-93*, pages 139–144.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Xiong, Y., Sadeh, N., and Sycara, K. (1992). Intelligent backtracking techniques for job-shop scheduling. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA*.

Zweben, M., Davis, E., Daun, B., and Deale, M. (1993). Informedness vs. computational cost of heuristics in iterative repair scheduling. In *Proceedings of IJCAI-93*, pages 1416–1422.