Using Sensitivity Analysis to Illustrate and Improve Schedule Robustness

Lisa A. Swenson and **James Little** and **Joseph Manning** and **Roman van der Krogt**
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{l.swenson | j.little | roman}@4c.ucc.ie; manning@cs.ucc.ie

Abstract

Robust schedules are important to industry for reasons, ultimately, of profit and maintaining competitiveness. We propose that sensitivity analysis can be used to improve schedule robustness. We illustrate this using two case studies, the classic bridge building problem and from real life, a semiconductor manufacturing problem. In both cases, we generate a predictive schedule with the performance objective of minimising makespan. Disruptions to the schedule are modelled respectively as duration extensions and release time delays. From a senstivity graph, we can formulate a hypothesis around how to introduce float into the original schedule so that it becomes more robust to the same changes with a possible, though not necessarily obligatory, extention to the manufacturing time.

Introduction

Finding robust schedules is a very hard problem. Not only do we have to find a schedule, but we also have to ensure that it does not break down or does not break down too much when disruptions occur. For constraint-based scheduling, one reference approach that can be used is the search for so-called super solutions (Hebrard, Hnich, and Walsh 2004) or weighted super solutions (Holland and O'Sullivan 2005). That is, we search for a schedule that is guaranteed to be repairable with a limited number of changes, if a disruption occurs without extending the makespan. Unfortunately, for all but small unrealistic problems, finding a super solution takes a prohibitive amount of time.

An alternative to finding a super solution is to find an optimal schedule, in terms of a desired objective, and to use the results of sensitivity analysis to determine and improve the schedule robustness. Sensitivity analysis is the study of the effect of uncertainty in model parameters and input variables on the output of a given model (Saltelli, Chan, and Scott 2000). Since a predictive schedule is a model for obtaining a performance objective, and because of unpredicted events, we can use sensitivity analysis to determine the impact of uncertainties on the predictive schedule and perhaps make the schedule more *robust* to those uncertainties.[1] This is not too far removed from the method employed in super solutions. With them, at each node of the search tree a value is assigned to a variable. The system then searches also for a solution in which that variable is perturbed by excluding that possible value. Our form of sensitivity analysis will make sequential perturbations to variables (durations and start times) of the schedule.

Note that sensitivity analysis in the scheduling context has been studied by several authors. For an overview see Hall and Posner (2004). However, in terms of robustness, these investigations deal with either bounding the change in performance objective for a given change in schedule inputs or with selecting a schedule from a set of optimal schedules, depending on schedule variable changes (Trystram, Penz, and Rapine 2000; Kolen et al. 1994; Jia and Ierapetritou 2004). Additionally, these sensitivity analysis investigations are tied to specific schedule scenarios. We propose to improve the robustness of a predictive schedule based on sensitivity analysis and, while recognising possible objective deterioration, to do so in a way that can be applied generically.

In this paper we present ongoing work in which we investigate using sensitivity analysis, via tornado graphs, to both measure and improve schedule robustness. Our conjecture is that we can use the information the sensitivity analysis provides to make small changes, such as inserting slack or swapping tasks, to make the schedule more robust. We propose an algorithm which takes the results of a sensitivity analysis to generate a new more robust schedule. We illustrate our premise using a case study in which we limit ourselves to "one-way" analysis; that is, investigating the effects of varying one parameter at a time. Multi-parameter analysis is left for future work.

The remainder of this paper is organised as follows. The next section introduces tornado graphs, the medium used to illustrate schedule robustness. This is followed by the discussion of an algorithm that uses the information provided by the sensitivity analysis (and demonstrated by the tornado graphs) to improve schedule robustness. This algorithm is then applied in a case study, involving real data from a semiconductor manufacturer. Finally, we discuss some of the research questions that this approach poses, and present a

[1] We say that a schedule is robust for a given objective under a certain rescheduling policy and for a given class of disruptions, if the objective value does not deteriorate when one of these disruptions happens and the rescheduling policy is applied to rectify it.

preliminary investigation into some of the answers to those questions.

Tornado Graphs

Communicating schedule robustness measures to industry has been a problem. Kempf et al. (2000) state that "a clear understanding of how the quality of a schedule is assessed is critical to the successful implementation of scheduling systems in real-world manufacturing environments." Additionally, experience with scheduling in the real world indicates that graphics are highly useful in communicating with industry (van der Krogt, Little, and Simonis 2009). For this reason, we have used *tornado graphs* to illustrate schedule robustness. A tornado graph is a type of bar chart (Eschenbach 1992), consisting of horizontal bars to the left and right of a central vertical line, as shown in Figure 1. Each bar on the left represents the amount of change in a specific model variable or parameter being investigated. Each bar to the right illustrates the effect on the model outcome of the corresponding change in the variable or parameter. The bars are ordered from top to bottom in decreasing magnitude of effect which gives the chart its characteristic "tornado" shape. Such graphs provide a richer description of the robustness of a schedule than a single number. At the same time, we can still obtain a single measure of robustness, if needed, by computing the sum of the lengths of the right-hand side bars.

To illustrate the use of these tornado graphs, we use the classic Bridge Building problem. This problem was introduced by (Bartusch 1983) and later accepted as a standard benchmark in the Constraint Programming community. A full description can be found in (van Hentenryck 1989). The problem involves the construction of a five segment bridge, in a minimum amount of time. To do this, we have to schedule 43 activities, 34 of which are allocated to the 7 available resources while the rest require no resource. The activities include excavations, making foundation piles, formwork, masonry work, positioning of the bearers, etc. If the contract for building the bridge includes a clause on when it should open, the builders would be very interested to know what the effect of a delay in one of the activities is on the overall project.

For example, consider a single activity a taking 10% more time than expected. We can study the impact of such an event by rescheduling the remainder of the activities (i.e. those that are scheduled to start at the same time or later as the offending task a) taking into account the longer duration for a, and examing the differences with the baseline schedule in which none of the activities has a longer than expected duration. Part of the tornado graph that presents the results of this analysis for each of the activities is given in Figure 1. On the left-hand side, we see the amount of time that each activity is extended by, on the right hand-side, we see the effect on the overall schedule. For example, we can see that when the task "*pstnBrr5*" takes 12 time units more than expected (10% of its duration of 120), the knock-on effect of this is that the schedule now completes 32 time units later. On the other hand, the schedule can absorb a 15 time unit increase to the "*fill1*" task (near the bottom of the graph) without any effect on the makespan. Since the tasks are ordered from largest to smallest impact, it is immediately clear which tasks are critical to the on-time completion of the bridge.

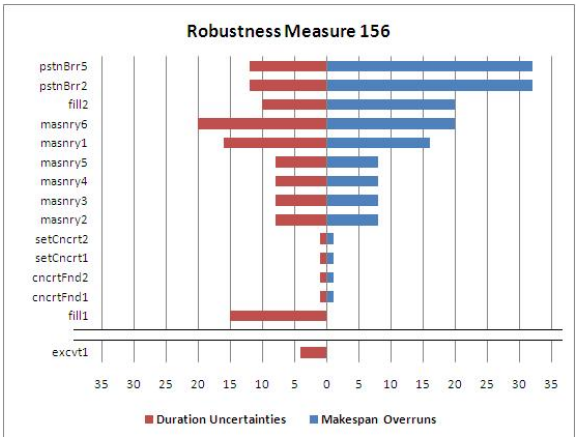

Figure 1: A tornado graph with its robustness measure

For the purposes of this paper, we extend tornado graphs to include a single numerical measure of robustness as the total area of schedule breakage; i.e., the total area of the bars on the right side of the graph. This allows for a numerical comparison of schedule robustness between schedules; in this case the number is 156.

Improving Schedule Robustness

The results presented in the previous paragraph are based on the default solution that Ilog OPL Studio returns when we solve the bridge problem. We know it has an optimal makespan, but the question is: can we improve upon the robustness of this schedule? It turns out that, in this case, we can. For example, one alternative schedule that has the same overall makespan has the robustness profile shown in Figure 2. As one can easily confirm, the impact of the delays is much less in this schedule than in the previous one. The reason for this is that the slack that is naturally available in the schedule is distributed differently. In the first schedule, the slack is concentrated in a few places, whereas in the second one, the slack is more distributed across the schedule and located around the worst offending activities. This observation gave rise to the following hypothesis: by identifying which activities are critical (i.e. cause the biggest impacts) and changing the schedule to include more slack around those activities, we can improve the robustness of schedules.

To investigate the hypothesis, we designed the following heuristic, as sketched in Algorithm 1. First, we obtain a base line schedule for the given set of tasks. Then, we simulate a delay for all tasks sequentially and identify, if any, which task a causes most impact on the objective of the schedule. If the impact is less than the given threshold, stop. Otherwise, we add as much float to task a as we can without impacting on the objective, with a maximum of x. Finally, we schedule all the tasks again and perform another sensitivity analysis.

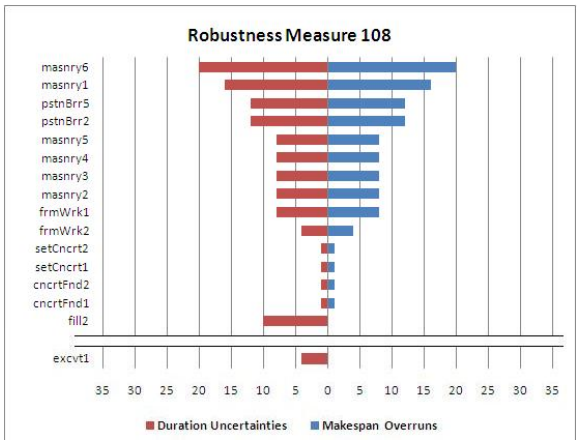

Figure 2: The tornado graph of an improved schedule

We used this heuristic in two different settings: that of the bridge building problem, and also on a real-world application: scheduling one of the steps in semiconductor manufacturing. We will first show the results on the real-world data, before we examine the bridge-building problem again in more detail.

Case Study : Semiconductor Manufacturing

We now discuss a case study, based on real-world data from a phase of semiconductor production. This phase is re-entrant as product leaves and returns to the same area, perhaps several times, during their manufacturing cycle (van der Krogt 2007). A constraint model built in ILOG OPL Scheduler was used to examine the robustness in terms of the makespan. Constraint-based scheduling is used both to create the baseline schedule and to reschedule, based on a single disruption (duration or start time). Tasks that had already been executed or were executing at the time of the disruption were fixed at their original times.

There are several variables to this scheduling problem: multiple machines, many possible operations, variable setup times and the possibility of doing two products on a single machine if they are undergoing the same operation. Additionally, there is an inter-operation duration, due to the re-entrant nature of the process. The inter-operation duration is dependent on the previous operation the product underwent and its next operation when it returns to the area. The inter-operation duration determines when the product enters the buffer for its next operation and is a major source of uncertainty. Thus, experiments were done using buffer entry delays as our modelled schedule disruptions and we examine the robustness in terms of makespan.

The Scheduling Algorithm

The plant runs around the clock. We arbitrarily used a 100-minute scheduling horizon with the primary objective of minimising makespan while at the same time maximising throughput by pairing tasks whenever possible. The following constraints were present:

Algorithm 1: Improve schedule through sensitivity analysis

input : $T = \{t_1, \ldots, t_n\}$ a set of tasks
τ a threshold for stopping
$\mathcal{S}$ a scheduling algorithm
$\mathcal{R}$ a rescheduling algorithm
Δ a function returning the change in objective value between two schedules
output: an improved schedule
begin
  while true **do**
    schedule $\leftarrow \mathcal{S}(T)$
    foreach $t_i \in T$ **do**
      let t_i' be the task t_i delayed
      schedule$' \leftarrow \mathcal{R}$(schedule, $T \setminus \{t_i\} \cup \{t_i'\}$)
      effect$[t_i] \leftarrow \Delta$(schedule, schedule$'$)
    let $a \leftarrow \text{argmax}_{t_i \in T}$ effect$[t_i]$
    if effect$[a] \leq \tau$ **then**
      return schedule
    else
      let a' be task a with added slack
      $T \leftarrow T \setminus \{a\} \cup \{a'\}$

1. The task has to finish within the scheduling horizon.
2. No task start time can begin before the wafer entry time in the buffer.
3. No task start time can begin before the previous operations on the wafer are completed.
4. No task start time can begin before the required setup time between operations.
5. Tasks paired on the same machine have to start at the same time.
6. No two tasks can be scheduled on the same port at the same time.
7. Tasks can be paired on the same machine if they have the same operation and arrive in the buffer within 30 minutes of each other.

Rescheduling with the original algorithm allowed tasks that had not yet begun execution to be swapped around to other machines. This was also considered realistic in that the process of loading a wafer on a machine is automated, so there is no issue with a task previously scheduled on one machine being swapped to another machine as long as the required setup time is accounted for in the schedule.

Sensitivity Analysis

Buffer entry delays of 5 and 10 minutes were modelled sequentially for each task. All tasks had the same likely magnitude of delay. This magnitude was large enough to cause some disruption to the schedule and in line with typical process durations (13-40 minutes) and setup times (1-9 minutes). The scheduled tasks were delayed, one at a time, and the rescheduling algorithm executed to determine the effect

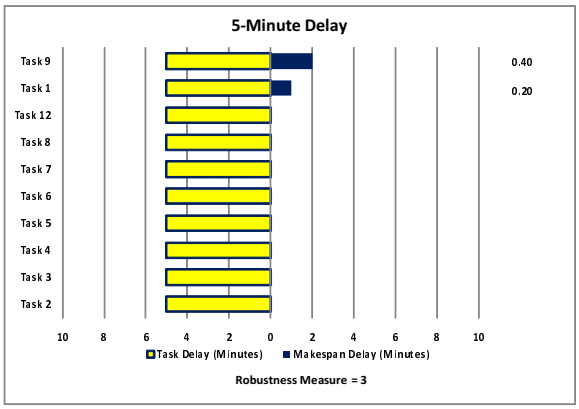

Figure 3: Tornado Graph for 5-Minute Delay

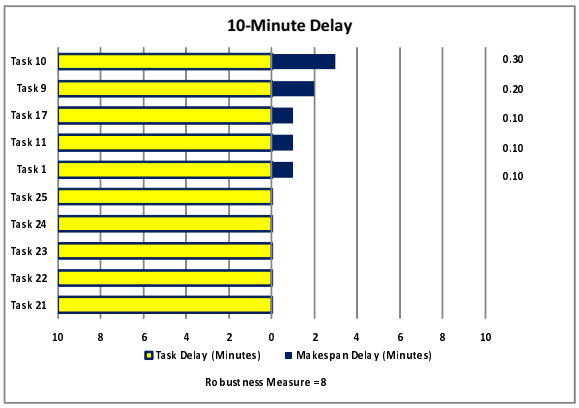

Figure 4: Tornado Graph for 10-Minute Delay

of the delay on the original predictive makespan of 99 minutes. A tornado graph was then created for each experiment.

Figures 3 and 4 are the tornado graphs showing the effects of a 5 and 10-minute delay in buffer arrival time for each of the tasks. Note that the figures have been truncated for clarity to show all tasks whose delay breaks the makespan plus some of those that do not. The rest of the tasks, which are not shown, do not break the makespan. Figure 3 shows that the predictive schedule is more robust to a 5-minute delay than a 10-minute delay as shown in Figure 4, as might be expected. Further the task with the biggest impact on makespan changed between 5 and 10 minutes.

Improving Robustness

The tornado graphs clearly illustrate the tasks whose buffer delays cause a disruption to the predictive schedule makespan. They also show which tasks have the bigger effect and the relative change in objective to a given delay. Consider again Figure 3. We used this figure to determine which tasks to investigate to see if we could improve the robustness to 5-minute delays in our original predictive schedule. We started with Task 9 as this task caused the maximum disruption of 2 minutes beyond our optimal makespan. Figure 5 shows this graph for the improved schedule. The new

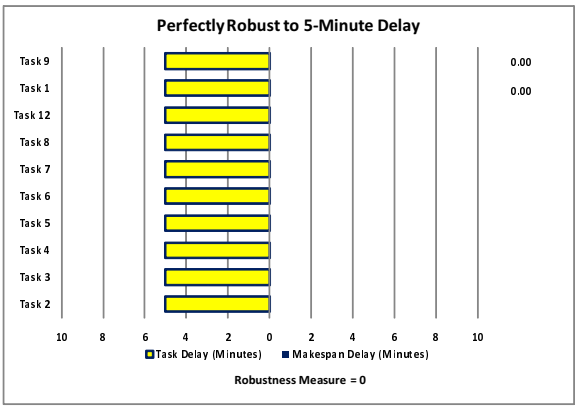

Figure 5: Tornado Graph for 5-Minute Delay, Improved Schedule

predictive schedule is perfectly robust to any one 5-minute delay and the makespan of 99 minutes remained the same as before so we improved our robustness without any sacrifice in optimality. The new schedule also, inadvertently, fixed the 1-minute disruption to the makespan caused by a 5-minute delay to Task 1. (Note that, as with the other tornado graphs, the other tasks not shown did not impact the schedule makespan when each was delayed by the specified amount.) We can consider this a "super solution" schedule where any delay of up to 5 minutes can be handled by having an alternative schedule with zero impact on the objective.

The same type of analysis was done for the 10-minute delay tornado graph shown in Figure 4. Based on that analysis, adding 10 minutes of slack to Task 10, greatly improved the original predictive schedule to 10-minute delays. Executing the rescheduling simulation again shows that the new schedule is robust to a 10-minute delay to all tasks except tasks 1, 11 and 17. Figure 6 is the tornado graph for 10-minute delays applied to the improved predictive schedule. Note that if we allowed the makespan of the improved predictive schedule to increase to 100 minutes from the original makespan of 99 minutes, we would have a perfectly robust schedule in terms of a 10-minute delay to any one task and a super solution. This would be a small trade-off in optimality for robustness in the face of delay.

Discussion

Having shown that the method is viable in a real-world setting, we now return to the bridge building example, to discuss some of the research questions that our approach poses. Firstly, we wanted to know if it is always best to allocate additional slack to the activity with the largest impact. Secondly, we were interested to see what happens if we allocate slack to the top n offenders, rather than a single one.

To investigate the first question, we took the 14 activities that showed an impact on the scheduling horizon in Figure 2. For each of those activities, we then formulated a scheduling problem in which that activity had its duration increased by 10%, solved it to optimality and did a sensitivity analysis. The problems that we generated can be considered the

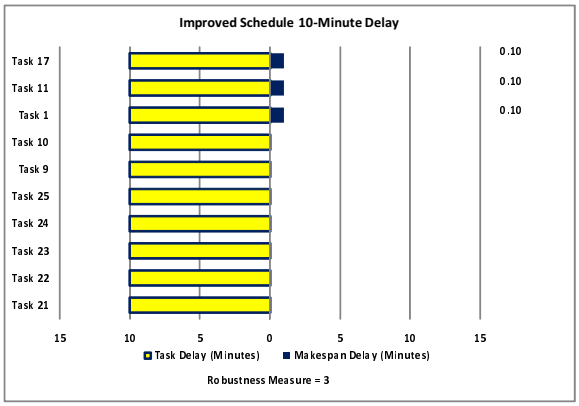

Figure 6: Tornado Graph for 10-Minute Delay, Improved Schedule

Table 1: Sensitivity analysis for each possible activity

break	Makespan	Robustness	# breaks	max break
20	1060	24	2	12
16	1056	68	6	20
12	1052	24	3	12
12	1040	108	14	20
8	1048	68	6	20
8	1048	68	6	20
8	1048	60	5	20
8	1048	84	7	20
8	1040	98	11	20
4	1040	132	15	20
1	1040	144	17	20
1	1040	98	11	20
1	1040	144	17	20
1	1040	98	11	20

most optimal schedule in which there is enough slack for that one activity to fully absorb its possible delay. (Notice that this means that the schedule may have to grow in length in order to fully provide the slack. This is different from our case study, where the makespan was kept fixed.) The results of this analysis can be found in Table 1. The table lists the size of the original break (i.e. the amount by which the makespan increases when this activity is delayed in the baseline schedule), followed by the results of the sensitivity analysis for the schedule in which that activity has additional slack. Indeed, this shows that the most robust schedule is the one in which the worst offender was repaired (only two activities that make the makespan increase, for a total of 24 units). However, there are several more good alternatives. The third row, for example shows an alternative that is only slightly worse (3 activities that break the makespan, with a total of 24 units of increase), but which is a lot better in terms of its makespan. Clearly, it is not always a clear-cut answer, and this poses an interesting question for future work.

We can also observe that assigning slack in the wrong places can be detrimental. For example, consider the second last row, in which the robustness measure increases from 108 to 144. Finally, we note that in general, allowing for a larger increase in makespan will lead to the best robustness measures, demonstrating the trade-off between optimality and robustness.

Table 2: Sensitivity analysis when allocating slack to multiple activities

n	Makespan	Robustness	# breaks	max break
1	1060	24	2	12
2	1076	24	2	12
3	1088	12	1	12
4	1088	12	1	12
5	1096	12	1	12
6	1100	10	1	10
7	1100	10	1	10

Table 3: Sensitivity analysis when allocating slack to multiple activities, dynamically

steps	Makespan	Robustness	# breaks	max break
1	1060	24	2	12
2	1072	12	1	12
3	1072	12	1	12
4	1072	12	1	12
5	1084	0	0	0

In response to the second question, we refer the reader to Tables 2 and 3. These show what happens when a schedule is created that includes slack for multiple activities. For the first table, we followed the same strategy as previously, except that we increased the duration of not one, but n activities (the n worst ones, $1 \leq n \leq 4$) when we generated the new schedule. As one can see, the robustness of the solution slowly improves, but not greatly, and at a large expense in optimality.

An alternative approach is to look at the sensitivity analysis at each step, and include slack for the worst offender in that case. The reasoning behind this is that when we alter the schedule to be more robust for the worst offender, the new worst offender is not necessarily the second-worst offender in the original analysis. Therefore, we expect this dynamic approach to be better than the previous, static, one. The results of this experiments are shown in Table 3. As one can see, after five steps we achieve a solution that is completely robust against any delays. Indeed, the solutions appear to be of a better quality than in the static approach, as the same robustness is achieved with a smaller decrease in optimality.

Conclusions and Future Work

In this paper, we introduced the idea of using sensitivity analysis to point us towards certain scheduling parameters which we can use to to build scheduling algorithms which introduce robustness in a trade-off with other objectives. A case study demonstrated that such an approach could be a viable alternative to producing robust schedules via other, more expensive ways such as the super solutions framework. Even a simple heuristic as presented here already improves

the robustness, and we hope to improve upon this with more complex heuristics.

We also introduced tornado graphs as a tool to communicate robustness to users. Most measures of robustness in the literature give a single number that tries to encompass the trade-off between schedule objective optimality and schedule robustness. For example, Leon, Wu, and Storer (1994) use a weighted linear combination of expected makespan and expected makespan delay to measure schedule robustness. Whereas, Carrillo and Daniels (1997) develop a statistical value, beta-robustness, that is the probability of a given schedule meeting a given minimum performance level. Additionally, Surico et al. (2006) develop a risk factor based on statistics for waiting times and maximum delays. Such single values indicating the robustness of each schedule communicates if one schedule is more robust than another but says little about how or what the schedule is sensitive to. We argue that the tornado graphs give a much more understandable picture for the users.

Our next step is to develop the algorithms further, through applying the technique to a variety of other scheduling problems. We already discussed some of the research questions open to us in the previous section. Furthermore, we want to apply statistical knowledge about the likelihood and magnitude of possible disruptions to the sensitivity analysis (where available). That is, to analyse only those disruptions that are likely to happen within a given confidence interval. Finally, the analysis could also be extended to a multi-way sensitivity analysis to model more than one type of disruption occurring within the schedule horizon.

Acknowledgments

The authors gratefully acknowledge support from Science Foundation Ireland under Grant number 08/RFP/CMS1711.

References

Bartusch, M. 1983. *Optimierung von Netsplänen mit Anordnungsbeiziehungen bei knappen Betriebsmitteln*. Ph.D. Dissertation, Fakultät für Matematik und Informatik, Universität Passau.

Carrillo, J. E., and Daniels, R. L. 1997. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions* 29:977–985.

Eschenbach, T. G. 1992. Spiderplots versus tornado diagrams for sensitivity analysis. *Interfaces* 22(6):40–46.

Hall, N. G., and Posner, M. E. 2004. Sensitivity analysis for scheduling problems. *Journal of Scheduling* 7:49–83.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Super solutions in constraint programming. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems, CP-AI-OR-04*, 157–172.

Holland, A., and O'Sullivan, B. 2005. Weighted super solutions for constraint programs. In *Proceedings of the 20th national conference on Artificial Intelligence - Volume 1*, 378–383. AAAI.

Jia, Z., and Ierapetritou, M. G. 2004. Short-term scheduling under uncertainty using MILP sensitivity analysis. *Industrial Engineering and Chemical Research* 43:3782–3791.

Kempf, K.; Uzsoy, R.; Smith, S.; and Gary, K. 2000. Evaluation and comparison of production schedules. *Computers in Industry* 42:203–220.

Kolen, A.; Kan, A. R.; Hoesel, C. V.; and Wagelmans, A. 1994. Sensitivity analysis of list scheduling heuristics. *Discrete Applied Mathematics* 55:145–162.

Leon, V. J.; Wu, S. D.; and Storer, R. H. 1994. Robustness measures and robust scheduling for job shops. *IIE Transactions* 26(5):32–43.

Saltelli, A.; Chan, K.; and Scott, E. M. 2000. *Sensitivity Analysis*. John Wiley and Sons, Ltd.

Surico, M.; Kaymak, U.; Naso, D.; and Dekker, R. 2006. Hybrid meta-heuristics for robust scheduling. Research Paper ERS-2006-018-LIS, Erasmus Research Institute of Management (ERIM).

Trystram, D.; Penz, B.; and Rapine, C. 2000. Sensitivity analysis of scheduling algorithms. *European Journal of Operational Research* 134:606–615.

van der Krogt, R.; Little, J.; and Simonis, H. 2009. Scheduling in the real world: Lessons learnt. In *Proceedings of the ICAPS 2009 Scheduling and Planning Applications Workshop*. ICAPS.

van der Krogt, R. 2007. Scheduling implant operations using constraint-based scheduling (abstract). In *Online Proceedings of the Third Workshop on Simulation in Manufacturing, Services and Logistics*.

van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. MIT Press.