

Using Tornado Graphs 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 first propose that tornado graphs give better insight into schedule robustness than a single numerical robustness measure. Secondly, we argue that this insight can be used to improve schedule robustness. We illustrate this using a case study from a semiconductor manufacturer to generate a predictive schedule with the performance objective of minimising makespan. Disruptions to the schedule are modelled as release time delays and the schedule robustness to those delays conveyed via tornado graphs. From the 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 while possibly extending 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. 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 will make sequential perturbations to variables (start times) of the schedule.

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. (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. For example, in the case study presented later in the paper, we create a predictive schedule with the objective of minimising the schedule makespan while at the same time maximising the number of tasks executed. The class of disruptions we consider are delays in task buffer entry times and our rescheduling policy consists of a full reschedule.)

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 so-called 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 initial 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 and explains the advantages of using tornado graphs to illustrate schedule robustness. This is followed by our case study using real-world data from a semiconductor manufacturing plant, a scheduling algorithm and sensitivity analysis via tornado

graphs to inform a new scheduling algorithm which will improve schedule robustness. Lastly, we give our conclusion and ideas for future work.

Tornado Graphs

A tornado graph is a type of bar chart (Eschenbach 1992). The graph consists 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.

The tornado graph in Figure 1 represents a one-way analysis on task duration uncertainties for a fictional schedule with 5 tasks. The data represented by the bars on the left are the “what if” values of the sensitivity analysis. That is, they are the schedule uncertainties under investigation. In this case, they are the uncertainty in task durations. Note that the magnitude of change in task duration being investigated varies with each task. This is because, ideally, we would study only the magnitude of changes most likely to happen.

The bars on the right represent the effect of the uncertainties on the desired performance objective. In this fictional scenario, the schedule objective is to minimise cost per unit of production. The magnitude of the bars on the right represent the increase in the expected cost per unit of production (decreases would be listed in brackets) as predicted by a rescheduling algorithm. Note that since this is a one-way analysis, rescheduling is executed for a single change in input variable. Thus each bar represents a single change in the named task duration and the effect of that change on the cost per unit of production.

We can see immediately from the graph that the duration increase in Task 2 has the most impact on the schedule objective while the duration increase in Task 5 has the least impact. That is, the schedule is most sensitive (least robust) in terms of Task 2 duration uncertainty and least sensitive (most robust) in terms of Task 5. This can be valuable information to start to investigate ways of improving schedule robustness. The figure also shows the values for the ratio of change in cost per unit to change in task duration added to the right of each bar.

Since the bars on the right side of a schedule tornado graph are a measure of how sensitive a deterministic schedule objective is to schedule uncertainties, the graph gives us a measure of schedule robustness. There are several possible numerical values we could take from the tornado graph as a robustness measure. For example, we could take the maximum change in schedule objective, which is 10 in Figure 1, or the total area of changes to objective which is 30 in the example. We could also formulate a linear, weighted combination of average objective value change and standard deviation for the changes. These values are all reasonable measures of robustness.

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

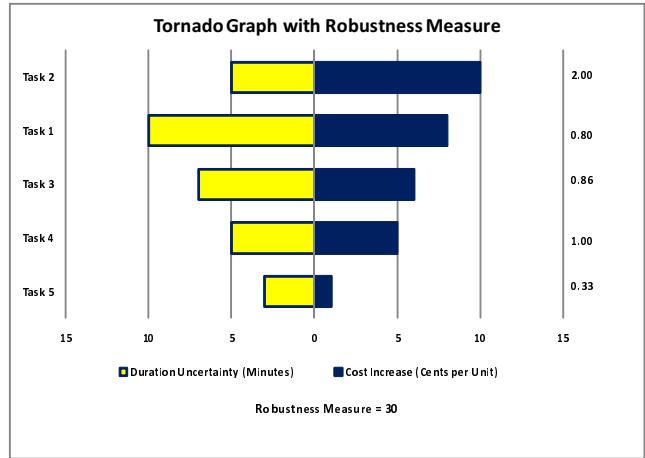


Figure 1: Tornado Graph With Robustness Measure

on the right side of the graph. This allows for a numerical comparison of schedule robustness between schedules. Figure 2 illustrates a tornado graph for a different schedule for the same fictional problem except this schedule is perfectly robust for the same duration increases. That is, the schedule absorbs each given disruption without any effect on its performance objective. Note that the single robustness measure for the schedule in Figure 1 is 30 while the robustness measure of the perfectly robust schedule in Figure 2 is 0.

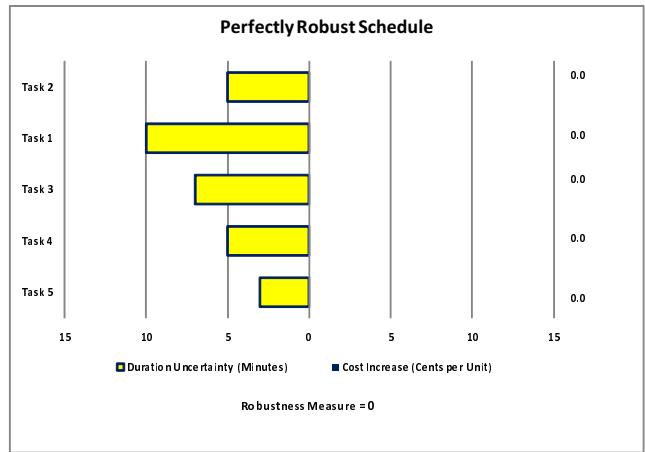


Figure 2: Perfectly Robust Schedule Tornado Graph

Advantages of Tornado Graphs

Sensitivity analysis is widely used in many disciplines such as, economics, environmental sciences and chemical research (Saltelli, Chan, and Scott 2000). Tornado graphs, as a tool of sensitivity analysis, would also have applications in these fields as a way to illustrate the results of a sensitivity analysis. We feel tornado graphs also have a significant advantage as a tool to both improve and communicate schedule robustness for the following reasons.

More Intuitive: 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). We argue that tornado graphs are more intuitive than a single number representing robustness while also providing a single measure of robustness if needed.

A tornado graph is an easy-to-understand tool for communicating a schedule’s ability to absorb disruption. Figures 1 and 2 very graphically demonstrate the different robustness levels of two different schedules to the same uncertainties.

A single value 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. 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. For example, the schedules related to Figures 1 and 3 both have a robustness measure of 30. Yet, the two schedules are not equal in which tasks they are sensitive to nor is each task having the same disruptive effect in both schedules. Note particularly in Figure 3 that the increases studied for tasks 4 and 5 have no effect on the schedule objective.

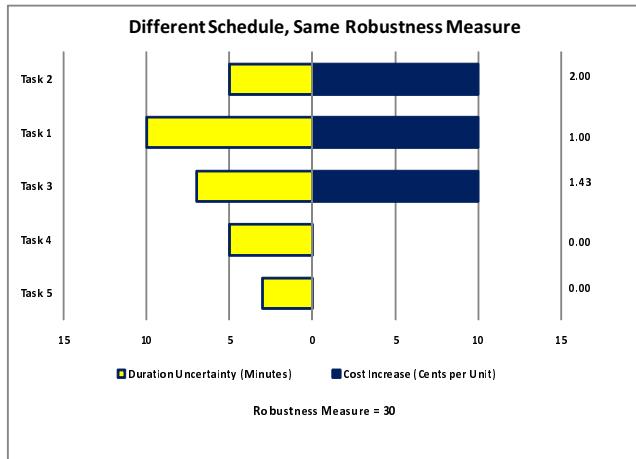


Figure 3: Different Schedule, Same Robustness Measure

It is also worth noting that Figures 1, 2 and 3 are a study of different schedules with the same uncertainties and the same units of measure. Thus, a comparison can be made between the schedules in terms of robustness to those particular un-

certainties.

More Generic: Tornado graphs can be applied to any scheduling or rescheduling algorithm and to any performance objective. You do not have to understand or implement any specific algorithms, as with many other robustness measures. As long as you have rules or an algorithm to create a schedule and to reschedule in the event of input variation, you can run “what if” scenarios to perform sensitivity analysis and create tornado graphs.

The generic nature of schedule tornado graphs has the advantage of allowing different scheduling/rescheduling algorithms to be easily compared. All that is required is that the compared schedules have the same performance objective, the same range of variation in input variables and that the same units are used for graphing. Again, Kempf et al. (2000) state that unless we can compare schedules generated by a given system to those generated by alternative systems in some objective and quantifiable way, we will lack a systematic framework for evaluating scheduling systems.

Additionally, many approaches to robust scheduling require that a company have statistical data on uncertainties in their scheduling inputs. A company may not have access to statistical data yet they may have experienced planners that have a sense of what goes wrong, how often and to what extent. Our approach lets the planner input simple uncertainties, which they might expect, to get an idea of the impact these uncertainties will have on their schedule.

In the case study in the next section, we use tornado graphs to improve our predictive schedule by adding slack selectively to those tasks likely to cause a problem in order to improve the predictive schedule’s robustness.

Improving Schedule Robustness - Case Study

Our experiment is 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).

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.

As stated earlier, we will limit ourselves to a one-way sensitivity analysis so we assume a single disruption during the schedule horizon for our model: the late arrival of a wafer in the buffer. Constraint-based scheduling is used both to create the baseline schedule and to reschedule based on the single disruption. We examine the robustness in terms of the 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. A constraint model, built in ILOG OPL Scheduler, was used with the following constraints:

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.

The same model was used to reschedule when a disruption occurred, with the exception that tasks that had already been executed or were executing at the time of the disruption were fixed at their original times.

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.

The Tornado Graphs

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

Figures 4 and 5 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 4 shows that the predictive schedule is more robust to a 5-minute delay than a 10-minute delay as shown in Figure 5, as might be expected. Further the task with the biggest impact on makespan changed between 5 and 10 minutes.

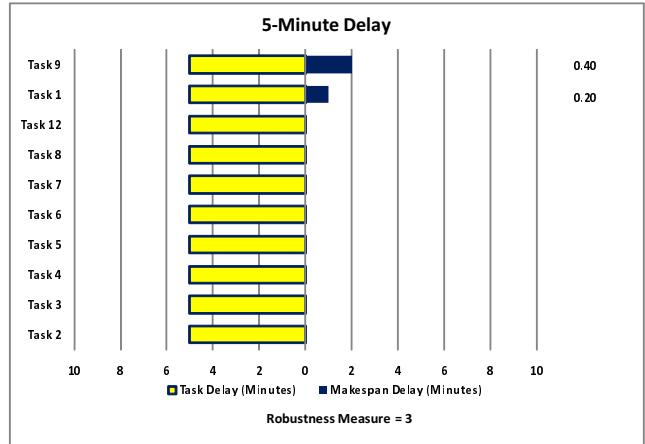


Figure 4: Tornado Graph for 5-Minute Delay

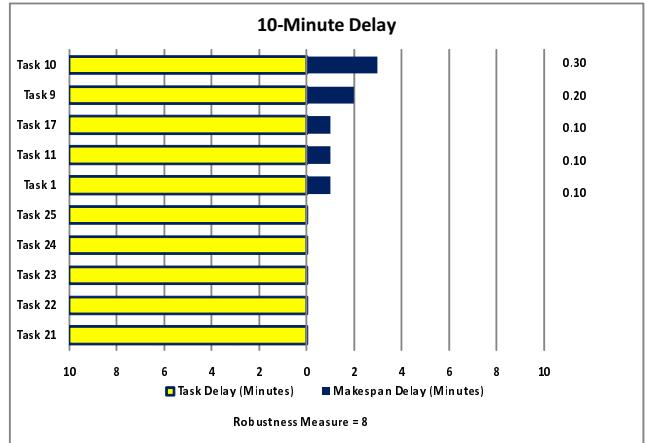


Figure 5: Tornado Graph for 10-Minute Delay

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. We argue that we can use this information to improve schedule robustness. Our algorithm for doing so is sketched in Algorithm 1. First, we obtain a base line schedule for the given set of tasks. Then, we simulate x minute release time delay for all tasks sequentially (we will investigate the cases $x = 5$ and $x = 10$). 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.

Algorithm 1: Improve schedule through Tornado Graph analysis

```

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

To illustrate this, consider Figure 4. 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 6 shows this graph for the improved schedule. The new 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

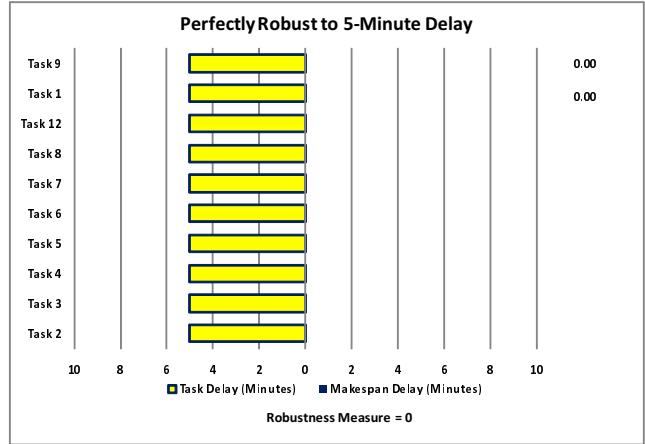


Figure 6: Tornado Graph for 5-Minute Delay, Improved Schedule

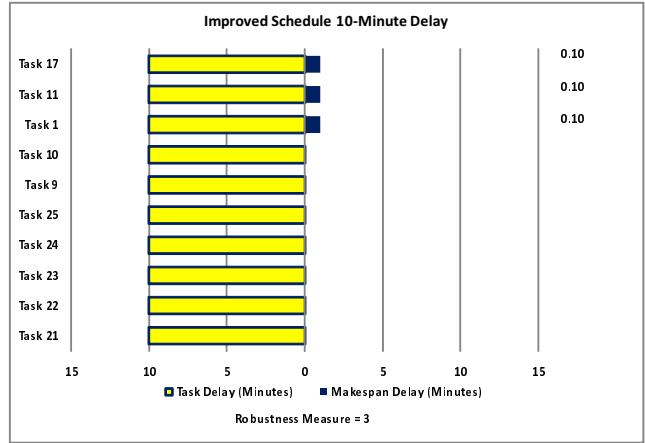


Figure 7: Tornado Graph for 10-Minute Delay, Improved Schedule

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 5. 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 7 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.

The case study demonstrates 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 al-

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

Conclusions and Future Work

In this paper, we introduced the idea of using tornado graphs to illustrate schedule robustness. We made the argument that tornado graphs are more intuitive for conveying schedule robustness than a single numerical robustness measure and, at the same time, also provide for a single numerical robustness measure if necessary. Tornado graphs point us towards certain scheduling parameters which we can use to build scheduling algorithms which introduce robustness in a trade-off with other objectives. Most importantly, we demonstrated this through the use of a case study on real-world data that these insights can be used to improve the robustness of the original predictive schedule.

Our next step is to develop the algorithms further, through applying the technique to a variety of other scheduling problems. Where available, we want to apply statistical knowledge about the likelihood and magnitude of possible disruptions to the sensitivity analysis. 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

- 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.
- 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*.