Optimization of operation sequences using constraint programming

Optimization of Operation Sequences using Constraint Programming

Nina Sundström * **Oskar Wigström** * **Petter Falkman** * **Bengt Lennartson** *

* *Automation Research Group,Department of Signals and Systems, Chalmers University of Technology, SE-412 96, Göteborg, Sweden, (e-mail: nina.sundstrom@chalmers.se)*

Abstract: In this paper, we connect the dots: design and optimization of production systems. A possible link between these two areas is a previously presented modeling language, sequence planner language (SPL). It has been demonstrated how relevant information can be extracted from production systems modeling applications, and converted into SPL. We show how the SPL model can be converted into a constraint programming model for optimization. Also, a useful abstraction concept, work equivalence, is introduced to enable alternative model formulations. A case study consisting of an aero engine structure assembly plant is presented, in which the efficiency of the resulting constraint programs is investigated. The formulations enabled by abstraction are shown to perform better than the standard formulation.

Keywords: Scheduling, Constraint programming, Flexible manufacturing systems, Discrete event systems, Multi-product production planning

1. INTRODUCTION

Scheduling in manufacturing and production industries tends to be quite complex. The production system itself consists of resources that enable various operations. Multiple products are to pass through the system, and each product requires a set of predefined operations to be performed in a specific order. This specific order is known as a product recipe, which can be described as a set of sequences of operations (SOPs). These can in turn be modeled using a modeling language called sequence planner language (SPL). Even though modeling tools can portray large systems, as the number of products increase so does the complexity of scheduling these operations. A framework for generating schedules for this type of systems is called job-shop scheduling which is a classical NP-hard problem.

It has been shown how SOPs can be automatically generated from manufacturing system modeling tools Magnusson et al. (2011). In this paper, we demonstrate how these SOPs can be transformed to constraint programming (CP) models. The OPL optimization programming language (OPL) Van Hentenryck (1999) is used for formulating the models. Additionally we introduce an abstraction, often applicable to flexible manufacturing systems, that enables some alternative CP modeling approaches. A case study consisting of an aero engine structure assembly plant is used to measure the CP model performance when implemented in the optimization software IBM ILOG Optimization Studio.

There are numerous languages available for describing sequential operational behaviour. In the industry, the most popular is perhaps Microsoft Excel, other popular more graphical representations are Gantt charts Kerzner (2003) and PERT charts Levin and Kirkpatrick (1966). Other alternatives used in software development include among others statecharts Harel (1987) (subset of UML). In Lennartson et al. (2010), a new language for expressing SOPs was presented, SPL, which holds a number of interesting properties. In contrast to the previously mentioned languages, it includes a formal definition of operations and SOPs as well as a full synchronous composition (FSC) operator Hoare (1978). The FSC operator is a necessary condition if one is to use supervisory control theory Ramadge and Wonham (1989) for supervisor synthesis, and operation states that simplify precondition and postcondition expressions for complex systems.

In flexible production systems, a large numbers of parts require processing. In the literature, this type of system falls into the category of job-shops Graves (1981) Pardalos and Resende (2002). In a flexible job-shop problem, a number of jobs, each consisting of a set of operations, are to be processed by a set of resources. Each operation can be processed by alternative or multiple resources. There are several variations, jobs can for example have precedence

⋆ This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance Production at Chalmers. The work has been supported by the EU-FP7 FLEXA project, Swedish Governmental Agency for Innovation Systems (VINNOVA) and Swedish Research Council (VR). The support is gratefully acknowledged.

constraints and machines can be related or have sequence-dependent setups.

Today, solving various job-shop formulations is still a large research topic. But in contrast to the early days of branch and bound techniques based on mixed integer programming Graves (1981), a myriad of competitive methods now exist. Modern approaches are often based on advanced heuristics and local search algorithms, e.g Nowicki and Smutnicki (2005). CP methods by themselves are relatively competitive while providing a more flexible approach Zhou (1996). In combination with heuristics they outperform even the best local search algorithms Grimes et al. (2009), see also for example Watson and Beck (2008) Malapert et al. (2009).

Most of the papers cited in the previous paragraph concern the general case and its extensions. Our focus is primarily on the transformation of automatically generated data describing a production system into a CP model. The development of new fast heuristics and propagation methods for this problem class is outside the scope of this paper. However, we have considered the impact of the model formulation and investigated which modeling approach is most suited for this specific problem category.

The paper is outlined as follows. Section 2 provides an introduction and characterization of flexible manufacturing systems considered in this paper. A brief guide to modeling using SPL is also included as well as the definition of work equivalence (WE) and WE abstraction. In Section 3 it is shown how an SPL model intuitively can be represented as a CP model. Finally, a case study of an aero engine structure assembly plant is presented in Section 4 followed by results in Section 5 and conclusions in Section 6.

2. MODELING

In a flexible manufacturing system, multiple products are to pass through the system according to a predefined set of SOPs called the product recipe. In the following subsections, flexible job-shop scheduling is introduced, as well as an SPL modeling tool, Sequence Planner (SP). Also, we define WE which is an abstraction method applicable to manufacturing systems.

2.1 The Flexible Job-Shop Problem

Job-shop problems concern the efficient utilization of resources while performing some predetermined tasks. More formally, given a set of jobs where each job consists of a set of operations. If the operations are to be processed in a specific order on specific machines, how can a set of resources be allocated as to achieve optimality. The optimization criterion could e.g. be to minimize the makespan, workloads of machines or waiting times, etc. Each operation has a given processing time and a resource required for processing. A flexible job-shop scheduling problem is a generalization of the job-shop problem. It enables an operation to be processed by any resource from a given set of resources. In this paper, a multi-resource system is considered where an operation may need several resources at the same time to be performed. The processing time for an operation is allowed to vary depending on the resource performing the operation.

An example of a classical job-shop problem is a hospital where patients are considered as jobs. Each patient follows a given route and is treated at different stations during their stay at the hospital. The resources in this case consist of nurses, doctors, rooms and equipment. A surgery may require at least one of each mentioned resource while a simple routine examination may only require a room and a doctor. In the multi-product system considered in this paper, several products pass through a manufacturing facility, each product following a route defined by its individual product recipe. All operations require several resources, for example a fixture and a robot in order to be moved. Some of the operations have a resource flexibility where different resources may perform the operation. There are also precedence conditions, for example when finished subparts are assembled into larger products.

2.2 Modeling using Sequence Planner

A modeling tool called SP has been developed by the Automation Group at Chalmers University of Technology Lennartson et al. (2010). It can be used for the definition and visualization of product recipes. SP uses an approach where operations can be modeled as self-contained with only relevant information on when and how an operation can execute. Sequences of operations based on e.g. different resources or products can be displayed by using a projection of operations that somehow relates to that resource or product. Recently, it has been shown that transport operations can be automatically generated from simulation software Magnusson et al. (2011). The next step is to also generate process operations based on the product recipe from external CAD software.

In SP it is possible to model straight, parallel, alternative and arbitrary sequences. In order to define relations between operations, pre- and postconditions can be used. These conditions have to be fulfilled before an operation can execute. For example, consider two operations in a straight sequence. When viewed as self-contained operations, a precondition is added to the latter operation stating that the previous operation has to be finished before the operation can start. Operations may also have pre- and postactions for booking and unbooking resources. When an operation requires one or more resources, a precondition for execution is that the necessary resources are available for booking. This implies that the preaction in this case, is the booking of the resources. A processing time for each operation can also be specified.

2.3 Work Equivalence Abstraction

When dealing with many products of the same type passing through a production system, abstraction can be used to reduce complexity. For example, within a number of product instances of the same product type, it is not necessary to explicitly keep track of each instance identity. This type of abstraction can be applied to more than just product instances of the same type. In this paper we introduce the concept of WE. In short, suppose that the SOPs or a subset of SOPs for a set of product types are the same, i.e. the SOPs have the same structure and resource dependence. If also the processing times are fully

or approximately equivalent, then this set of product types can be considered to hold WE or weak WE respectively.

Note that if two products hold weak WE, i.e. similar processing times, then the longest processing time can be relaxed to the shorter, and as such make the products WE. With this relaxation, solving the resulting optimization problem will yield a lower bound for the optimization problem. Reversely, the shortest execution time can be regarded as equivalent to the longer in order to derive an upper bound. In this paper we consider problems where the occurrence of WE is common.

The reason for introducing WE is that it enables abstraction of product types, permitting favourable changes to the optimization model. The WE abstraction works as follows, if a set of product types are WE, these can be regarded as consisting of one meta-product type. Each meta-product instance can represent any one of the WE products. Thus, if the processing order of the meta-product instances are constrained, no information is lost as each instance still represents any product. Suppose a different product type requires one of the WE products in order to start its execution. Then it is enough to know that one meta-product is available for processing. This 'buffer' of available meta-products can be kept track of via a time variable. Note that which explicit product type one specific meta-product instance actually represents, may be given implicitly by future operations.

3. CONSTRAINT PROGRAMMING

The origin of CP lies in the artificial intelligence and computer science communities and can be traced back to the constraint satisfaction problems (CSPs) studied in the 1970s Pinedo (2005). A CSP entails finding a feasible set of decision variables subject to a number of constraints, i.e. assigning values to variables that satisfy all constraints Rossi et al. (2006). During the last decades, CP has evolved into solving optimization problems, that is finding the solution in a feasible set that minimizes or maximizes a given objective function. The constraints may be of various different types; linear, nonlinear, logical, cardinal and global. This makes modeling problems using CP much more flexible compared to operations research, where only linear and integer constraints may be used.

3.1 Scheduling using Constraint Programming

In this paper, IBM ILOG Optimization Studio is used for implementing the optimization models. The software is an OPL integrated development environment containing among others, a CP solution engine. OPL is a language designed to model and solve optimizations problems using both CP techniques and mathematical programming approaches.

The decision variables for CP scheduling problems using OPL are intervals. Each interval represents an operation and is characterized by a start value, end value and size. Examples of constraints that are used in OPL to describe the structure of an SOP are e.g. *endBeforeStart()* and *alternative()*. The former states that the start of one interval has to be greater than the end of another interval. The latter models an exclusive alternative between different intervals. Another useful function in OPL is *cumulFunction* which can be used for the resource allocation in a job-shop scheduling problem. This function is incremented as a resource is booked and decremented when the resource is released, acting as a pulse function.

If a resource has to be booked for the duration of several operation intervals a pulse function may not be used. Instead, an OPL expression *stepAtStart()* can be used to increment the cumulative function at the start of an operation. In the same way, *stepAtEnd()* can be used to decrement the resource at the end of an operation. An upper bound for the *cumulFunction* corresponds to the capacity of the resource.

3.2 Mapping of Operation Sequences

As previously mentioned, operations in CP using OPL are represented by interval decision variables. The length of the intervals equals the processing time of each operation. The mapping of the SOPs depicted in Fig. 1 to CP will be described in the following paragraphs. An interval variable O_1 is initiated as

1: interval O_1 size(O_1^{min},O_1^{max})

where O_1^{min} is the minimum execution time and O_1^{max} is some sufficiently large constant. Serial execution of two operations, O_1 and O_2 can be described in two ways. Either by

1: endBeforeStart(O_1,O_2)

which allows for a segment of time between the two operations when nothing is performed. However, if a resource is to be booked during O_1, and until O_2 starts it can be convenient to use

1: endAtStart(O_1,O_2)

This constraint will ensure that O_1 ends as O_2 starts and thus guarantee that the resource is booked for the necessary period. Two operations, O_2 and O_3, executing in parallel without mutual dependence can be modeled using two sequential constraints. If O_1 is an operation preceding the two parallel operations, the system can be modeled by

1: endBeforeStart(O_1,O_2)
2: endBeforeStart(O_1,O_3)

If an SOP contains alternative operations, the *alternative*() constraint can be used. Suppose either O_5 or O_6 is to be executed after the preceding interval O_4. The following code will ensure the correct behaviour.

1: interval O_i[5..6] optional size(O_i^{min},O_i^{max})
2: endBeforeStart(O_4, D)
3: alternative($D, \{O_5, O_6\}$)

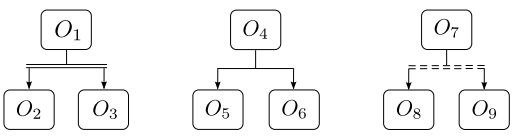


Fig. 1. Graphical representations of parallel, alternative and arbitrary sequences.

First, O_5 and O_6 are initiated as optional, i.e. neither of them have to be executed. Then, D , a dummy interval is constrained to start after O_4. The *alternative*() constraint then states that if D is executed, it will start and end with one of O_5 and O_6. This will force one of these optional intervals to be executed.

An arbitrary sequence is when two operations are both to be executed, but during different time intervals. This behaviour can also be interpreted as a parallel sequence and a dummy resource that mutually excludes the operations. The necessary constraints for mutual exclusion of two operations O_8 and O_9 are the following

1: endBeforeStart(O_7,O_8)
2: endBeforeStart(O_7,O_9)
3: cumulFunction C = pulse(O_8)+pulse(O_9)
4: $C \leq 1$

A *cummulFunction* C is used to represent a dummy resource. The *cummulFunction* is a time dependent variable which on row 4 is constrained to a maximum value of 1. A *pulse* is a timed expression which attains value 1 as its target operation executes. As a result, with C defined as the sum of two pulses on row 3, the two operations cannot execute simultaneously without violating the upper bound of C. In other words, O_8 and O_9 mutually exclude each other. However, the order in which they execute is not specified.

Pre- and/or post conditions may be added to operations. These define relations to other operations, e.g. an operation O_{11} cannot execute until another operation O_{10} has finished. This may be expressed as a precondition O_{10}^f for operation O_{11}. Expressed in OPL, simply *endBeforeStart*() could be used. The resource booking/releasing is handled through pre- and/or postactions. A preaction may be added to operation O_{11} to book a resource R. In the same way, a postaction may be added to release the resource. The corresponding pre- and postaction added to operation O_{11} are R^+ and R^-. As mentioned earlier, the resource booking/releasing can be modeled by cumulative functions. In this case a *cumulFunction* R is defined by $\sum_i pulse(O_i)$, where i are the indices of all operations that require R during execution.

3.3 Implementing Work Equivalence

From a CP modeling perspective, WE abstraction implies two things. First, the sequential order of meta-product instances can be explicitly defined. That is, for each set of WE products or WE subsets of SOPs, choose any arbitrary ordering and constrain the set of WE jobs to start in this predetermined order. In the case study it is shown that this is beneficial for large problems. The reason for this is a concept known as symmetry. Symmetry appears when multiple combinations result in the same solution. Take the hospital scheduling problem for example. Consider the case when two doctors with the same qualifications are to be scheduled. Suppose two schedules for the doctors are proposed, then it does not matter which doctor gets which schedule. In the same way, product instances of the same product type are also prone to symmetry. To remove this, they are constrained to be started in a certain order. What WE abstraction does, is to expose hidden symmetry in the problem and remove it. Weak WE that can be relaxed to WE can be thought of as almost symmetric elements.

Second, if there are preconditions that require some meta-product to be completed. Then the number of finished meta-products can be modeled by cumulative resource variables that are decremented during the mentioned preconditions. In practice, there are products which have precedence constraints on the completion of the WE jobs, create a cumulative function that represents the number of finished WE jobs. Have each job increment the cumulative function with a step at completion. Also, make the precedence constrained job decrement the cumulative function, as to indicate the consumption of a buffer element. If one job requires several other jobs to be started, multiple cumulative functions may have to be decremented. Using this abstraction, the constraints binding one product to another via a precondition is somewhat loosened. This implies that the size of the search space is actually increased. But in this particular case this leads to the local neighborhood of each solution to be expanded. This in turn will result in local search methods finding more solutions.

As the constrained order and the cumulative buffers abstraction methods work in two quite opposite directions, we found it best to apply only one method at a time. During which conditions it is advantageous to apply each abstraction method is discussed in the result section.

4. CASE STUDY

For our case study we have considered a manufacturing facility for aero engine structures, to be more specific the turbine exhaust case. Traditionally this structure has been delivered as one big piece of casting which has then been machined in several steps. The manufacturing process is very robust and several steps may be performed in the same station by using multi-purpose machines. However, a major drawback is that only a few suppliers in the world can deliver these big pieces of casting. The aero engine industries are therefore looking into the possibility of using automation in their manufacturing processes. They study possible methods to divide the structure into smaller parts which are automatically assembled to sub-assemblies which are further assembled to the final structure. Much in the same way as processes in the automotive industry where automation is used to a great extent.

4.1 Process Description

The manufacturing steps studied in this paper, are the processes that refine smaller parts before they are assembled, as well as the assembly operations of the smaller parts to the larger parts, called segments. The manufacturing facility contains tables for fixating and unfixating the parts, robots for transporting the parts, machines for milling, washing, deburring, measuring and welding. The parts and segments are attached to fixtures throughout all operations. There are 8 types of segments, each consisting of either 3 or 4 smaller parts. Additionally, there are 14 different types of parts. With WE, the parts/segments that use the same type of fixture and have the same processing time are bundled to one meta-product type. This results in a total of 5 part types and 2 segment types. The product

recipe for all part types except one is given by

Fixate → Milling → Unfixate → Washing → Deburring → Washing

The product recipe for the last part type is given by the last three operations in the above recipe. The product recipes for the segment types are given by

Fixate → Measuring → Welding → Measuring → Unfixate

There are two robots in the manufacturing cell that performs the necessary transportation of products between operations. The segment recipe only uses the first robot while the parts utilize both robots for transport. The characteristics of the considered system are given below.

- 21 resources, 49 parts, 13 segments, 787 operations
- Alternative transportation paths due to buffers.
- In the product recipes some operations are repeated, which results in parts returning to a workstation previously visited.
- Parts always have one ore more resources booked throughout their SOPs.

The analyzed manufacturing system has two identical parallel milling machines. Hence, if we apply WE on the machines, they can be viewed as one resource with capacity 2. Due to buffers, parts may before and after the milling operation be transported either to a buffer or the next workstation according to their product recipe. In the washing and the measuring workstation parts may come from an upstream or downstream workstation. For example, after the parts have been washed, a deburring operation will take place in a downstream workstation. Later, the parts will be transported back to the washing machine for further processing.

The SOPs for each part are modeled as interconnected operations, i.e. a succeeding operation starts at the end of a preceding operation. The lower bound of the operation interval equals the processing time of the operation. The upper bound is set to a large value. The processing time to e.g. fixate a part equals 2 minutes. However, after the operation has been performed the part may still occupy the loading table. A reason to why the part is not transported to the next workstation might be that the robot performing the transport is busy. Therefore, the operation interval may be longer than the actual processing time.

The objective is to minimize the makespan of the production of 13 segments, i.e. a total of 49 parts. This implies that we would like to minimize the largest completion time of the last operation for all segments. As mentioned earlier, there are two different types of segments. For the engine structure to be produced in this case study, there are 3 segments of the first type and 10 of the second type. We have investigated different ratios between the segment types including varying the total number of segments. Three different modeling approaches have been analyzed in order to compare which constraint program that is more efficient. These approaches are based on the abstraction methods described in Section 3.3. The different approaches are presented in the following section.

4.2 Modeling Approaches

As previously mentioned, WE can be used for two types of abstraction. We have compared these two types with 'out of the box' CP scheduling methodology, which we will simply refer to as the normal model. The two WE modified models are the constrained order model and cumulative buffers model. All models have identical procedures for resource booking and alternative operations. The latter behaviour originating from the buffers in the production system. The following paragraphs describe the three methods.

Normal The normal approach does not constrain the order between operations for parts of the same type, i.e. all parts and segments are free to start in any order.

Constrained Order For the constrained order approach, all meta-product instances are constrained to start in a specific order, i.e. all parts as well as segments of each meta-product type have a constrained order.

Cumulative Buffers No constraints applied to product order. Cumulative functions are used to model buffers of completed meta-products, i.e. the parts and segments are only connected via the cumulative buffers.

The performance measures of each method is presented in the following section.

5. RESULTS

In the previous section a manufacturing facility for aero engine structures was described. One complete structure requires 13 segments, 3 type-A and 10 type-B segments. Besides scheduling the specific combination of segments mentioned, 3/10 in short, we have run the algorithm for a number of other combinations. All optimization was run on a Windows 7 64-bit system with a 2.66 [GHz] Intel Core2 Quad CPU and 4 [GB] of RAM.

Fig. 2(a) shows the resulting duality gap for a number of segment combinations after 5 minutes of algorithm execution. The resulting gap after 25 minutes of optimization is depicted in Fig. 2(b). The duality gap is defined as $(Cost - LB)/LB$, where LB is a crude lower bound derived from the assumption that the welding station acts as a bottleneck. In reality, the asynchronous output of the milling machines and washing machine causes further delay. The segment combinations have been ordered by total number of parts in increasing order from left to right. The number of parts for the five different combinations are $\{33, 38, 41, 44, 49\}$ respectively. The white bar represents the cumulative buffers model, the black bar is the constrained order model and the gray bar shows the result from the normal model. Both WE modified models show better results after both 5 and 25 minutes. The constrained order model yields better results in the short run but looking at Fig. 2(b) shows that the cumulative buffers model performs better in the long run.

Fig. 3 shows the progression of the three algorithms for the 3/10 case. The dashed curve represents the normal model and is the worst of the three at all times. The dotted curve shows the cumulative buffers model and the solid is the constrained order model. Up to about 12 minutes into

the optimization, the constrained order model shows much better results, but in the long run it is outperformed by the cumulative buffers model.

6. CONCLUSION

This paper shows how sequence of operations can be converted into constraint programming models. These models can be used for scheduling of operations. We also present an abstraction method, work equivalence, which allows modification of the constraint programming model. A case study of a multi-product production system is presented. We evaluate various product configurations to measure the performance of constraint programming and work equivalence. Three different approaches were considered. A normal unmodified model was outperformed by two work equivalence models. One of the latter performed better in the long run, while the other could produce good results in short time. We hope to feed back the resulting time-based schedules to sequence planner and generate event-based control policies.

REFERENCES

Graves, S. (1981). A review of production scheduling. *Operations Research*, 29(4).

Grimes, D., Hebrard, E., and Malapert, A. (2009). Closing the open shop: contradicting conventional wisdom. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, 400–408. Springer-Verlag.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8, 231–274.

Hoare, C.A.R. (1978). Communicating sequential processes. *Commun. ACM*, 21, 666–677.

Kerzner, H. (2003). *Project management: A systems approach to planning, scheduling and controlling*. Wiley.

Lennartson, B., Bengtsson, K., Yuan, C., Andersson, K., Fabian, M., Falkman, P., and Åkesson, K. (2010). Sequence planning for integrated product, process and automation design. *IEEE Transactions on Automation Science and Engineering*, 7, 791–802.

Levin, R. and Kirkpatrick, C. (1966). *Planning and control with PERT/CPM*. New York, McGraw-Hill.

Magnusson, P., Sundström, N., Bengtsson, K., Lennartson, B., Falkman, P., and Fabian, M. (2011). Planning transport sequences for flexible manufacturing systems. In *Preprints of 18th World Congress of the International Federation of Automatic Control*.

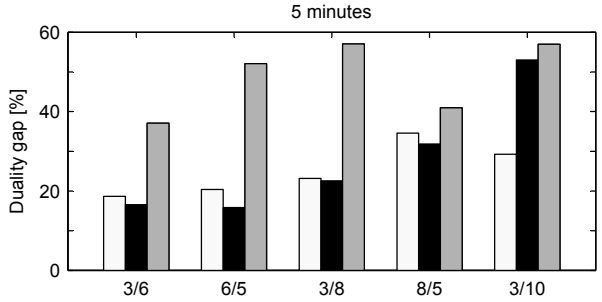
Malapert, A., Cambazard, H., Gueret, C., Jussien, N., Langevin, A., and Rousseau, L.M. (2009). An optimal constraint programming approach to solve the open-shop problem. Technical Report CIRRELT-2009-25.

Nowicki, E. and Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8, 145–159.

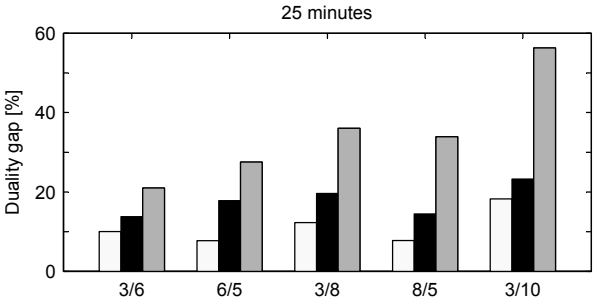
Pardalos, P. and Resende, M. (2002). *Handbook of applied optimization*. Oxford University Press.

Pinedo, M. (2005). *Planning and scheduling in manufacturing and services*. Number v. 1 in Springer series in operations research. Springer.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77, 81–98.



(a) The duality gap after 5 minutes of optimization.



(b) The duality gap after 25 minutes of optimization.

Fig. 2. The resulting duality gap after 5 minutes (a) and 25 minutes (b) of optimization. White bar shows cumulative buffers model, black bar is constrained order model and gray bar represents normal model.

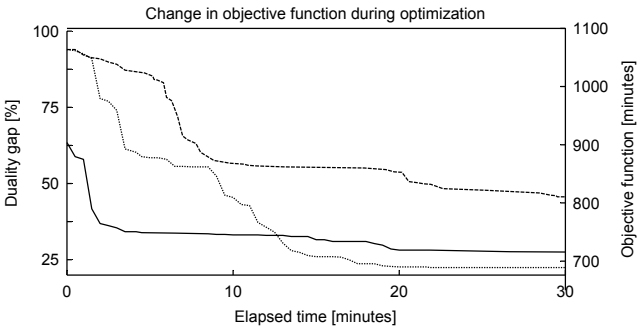


Fig. 3. Time/Objective plot for the case study (3/10). The normal model (dashed curve) gives the worst results. The constrained order model (solid curve) finds a low result fast while the cumulative buffers (dotted curve) achieves the best result in the long run.

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Van Hentenryck, P. (1999). *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA.

Watson, J.P. and Beck, J.C. (2008). A hybrid constraint programming/local search approach to the job-shop scheduling problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 5015, 263–277.

Zhou, J. (1996). A constraint program for solving the job-shop problem. In *Second International Conference on Principles and Practice of Constraint Programming (CP'96)*, 150. Springer Verlag.