Approximating a Global Objective by Solving Repeated Sub-problems for an Oven Scheduling Problem

Helmut Simonis
Insight SFI Centre for Data Analytics, School of Computer Science and Information Technology, University College Cork, Cork, Ireland

Abstract

In this paper we describe results for an oven scheduling problem studied during the European ASSISTANT project. This is a multi-stage scheduling problem arising in the production of rotor assemblies for compressors, provided by one of the industrial partners in the consortium. The main resource type is a set of identical ovens, which are used to heat-treat components in different ways. The process for one product may require multiple consecutive steps using these ovens, with specific temperature and process requirements at each step. Multiple tasks of different orders can be processed together in the same oven, if the temperature and process parameters for the tasks are identical. Processing multiple tasks together is more energy efficient, but typically forces some tasks to wait until all scheduled items are available, possibly impacting product quality and creating delays for the orders. The main difference to the oven scheduling problem studied in the literature is that we are not just trying to find an optimal solution to the short-term, detailed scheduling problem, but rather are interested in how selecting different parameters and constraints for the short-term scheduling problem affects the overall long-term, global objective of minimizing energy use, while maintaining the quality of products. Turning ovens off and then on again is considered bad for energy and maintenance reasons, we therefore try to minimize the number of shutdown events over the full planning horizon, while dealing with demand fluctuations over time. Information about jobs to be scheduled is only available within a limited time horizon, we therefore cannot solve the overall problem as one global optimization problem. Results indicate that we obtain a good overall schedule with a simple detailed scheduling model.

2012 ACM Subject Classification Applied computing → Industry and manufacturing

Keywords and phrases scheduling, global objective, oven scheduling, constraint programming

Funding This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 at Insight the SFI Research Centre for Data Analytics at UCC, which is co-funded under the European Regional Development Fund. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

1 Introduction

Scheduling is one of the most successful application areas of Constraint Programming (CP) [18, 4, 24, 11, 2, 3, 12, 5, 27] for over thirty years [26, 23, 21, 25, 6, 9, 10], with both academic benchmarks [14, 13] and real-world scheduling problems [7] being solved with CP over the last years. For these problems, one tries to find a solution of assigning start and end dates as well as resources to jobs and tasks, while respecting temporal constraints between the tasks, and overall resource constraints which control which tasks can run at the same time [8]. An important characteristic of these problems is that we are not only interested in finding a feasible solution, but rather search for a solution which "optimizes" some objective. For most large-scale problems, finding (and proving) an actual optimal solution is not feasible, so that we are searching for "good" solutions that are found in a limited amount of time. While there are literally hundreds of papers that consider specific constraints and propagation methods for solving scheduling problems, there only is rather limited discussion of how to select an objective function for a specific problem, and how solving snapshots of a scheduling problem at specific times helps with finding high quality solutions over time. In most academic problems, each instance is considered on its own. We ignore that the plant is currently used to implement yesterday's schedule, which limits at which time resources for a new schedule will become available. We ignore that tomorrow (or at another timepoint in the future) we will run the scheduling process again, and decisions we take today can have a negative impact on the overall plant efficiency over time.

Many scheduling applications also choose an objective function that does not directly express the quality of a solution. For example in many production scheduling problems we try to minimize the overall schedule end. While this is intuitive for project scheduling problems, say building a ship, it is less clear why this leads to a good solution over time for a factory. We do not plan to shut down the plant once the generated schedule has been run, but rather want to continue production in the next time period.

On the other hand, we may not have complete (or even any) information about the jobs that will need to be run at a later time, either since customers have not yet ordered the items to be made, or we do not know exactly when the current schedule will finish due to breakdowns or delays in the current schedule. Creating a schedule from this partial information may not be useful in planning the future actions required.

In this paper we try to consider these choices for a a relatively simple, but non-standard oven scheduling problem. We consider a manufacturing problem where at each time point we only see tasks arriving in the immediate future, while decisions we are taking now may limit the choices that are available in the future. The overall objective is to both finish jobs without a delay, which affects product quality, and at the same time reduce the overall energy consumption by sharing resources between tasks where possible. In particular we are concerned with using the correct number of oven resources over time, as the energy cost of heating up or maintaining oven temperature while not in use is significant.

Our main contributions are a definition of the overall problem in terms of short-term and long-term sub-problems, defining a simple constraint model to deal with the oven specific constraints, and evaluating test scenarios to find the best choices for parameters of the short term problem that lead to high-quality solutions over a longer planning horizon.

The constraints of oven scheduling make it a rather difficult problem to express with CP, the only previous solution for such a problem is described in [16, 17]. Multiple tasks from different jobs can use the same oven together, but only if their manufacturing profiles match, and they start and end their run together. This is not easy to express with the well known global disjunctive or cumulative [1] resource constraints. In their model, they introduce two separate sets of decision variables, one expressing the sequence of tasks on the oven resource, and another to express the temporal constraints for the manufacturing tasks. Element constraints link these models together. The overall model is quite specific to the CP-Optimizer [15] tool that is used, and heavily relies on specific constraints available in CP-Optimizer. In contrast, our proposed model avoids global constraints, and expresses the problem with disjunctions of primitive constraints. This model can be easily expressed in MiniZinc [19], and is well suited to mixed CP and SAT solvers like Chuffed [22] and OR-Tools [20], but also works quite well for MIP Solvers like Cplex.

The paper is structured as follows: In the next Section 2, we define the overall problem, introduce the short and long horizon sub-problems, and define some overall key performance indicators. In Section 3 we detail the model for the short horizon problem, which we solve repeatedly at consecutive time points. We then look at solving the short term (Section 4) and long term (Section 5) problems. We evaluate the approach based on different scenarios in Section 6, and conclude in Section 7.

2 Problem Definition

We provide a more abstract problem definition than would be required by the specific application problem to generalize the problem to handle use cases from different industries.

The problem consists of scheduling a set of jobs on a set of resources, finding the best compromise between conflicting objectives. Each job consists of a series of tasks in sequential stages, which need to be performed in order. Each task has a given duration, which does not depend on the resource allocation, but depends on the product made. Each task can run on any of the available machines, with no preference for specific machines. Different from traditional flow-shop problems, there is a single resource pool, the tasks of all stages compete for the same machines. An initial release date is given for each job, which is defined as the end of the previous manufacturing operation, which is determined by another scheduling system. If a task is waiting to be executed, it incurs a waiting cost. This is related to product quality, and a maximal allowed waiting time is given as input data.

In one variant of the problem, we assume a traditional machine choice model, each task exclusively occupies the machine on which it is running, tasks run continuously for the allocated duration, and tasks cannot be interrupted.

It is possible, but not required, that the tasks of a job are performed on the same machine.

In a more problem specific variant, we also consider that each task has a specific manufacturing process type. The type determines the duration of the task, and tasks of different stages always have different types. Two tasks of different orders with the same type can be processed together on the same machine, starting and ending at the same time. This not only increases the throughput of the factory, it also reduces the energy used, as for a given order set the machines are running for a shorter period. On the other hand, it rarely occurs that two jobs have the same release date, in order to process two tasks together, one of them has to wait until the second task is available for processing. This increases the waiting time, but on the other hand the resource will be released again sooner, so that later jobs may not have to wait for a machine to run on, potentially reducing subsequent waiting times. Running two tasks together on the same resource is called *task stacking*. The chances for applying stacking strongly depend on the properties of the order set, so that in some situations the traditional machine choice model may not incur much of an overhead. It is in principle possible to stack more than two tasks, but the current data do not allow this.

Ideally, machines are used for continuous runs, where some machines are used with only short interruptions, as this again reduces energy consumption. The gap between two consecutive tasks should be less than two hours, this allows to save energy over a case where a new machine is brought on-line. As the number of jobs can change significantly over time, we cannot always work with just a few machines. Also, note that if we delay a task until a currently running machine becomes available, we again incur a waiting time, that we can avoid if we use a currently unused machine.

2.1 Short Horizon

At a given timepoint, only part of the complete order set is visible, and we have to make our scheduling decision based on the available information only. In our industrial use case, a job becomes visible if it has started on the previous step of the manufacturing. An estimate of the end of that production step is used to define the release date for the job. We may decide not to use all orders that are currently visible, but that will only become available at a later time, and restrict ourselves to a fixed lookahead horizon in order to reduce the size of the problem to be solved.

We also have information about all tasks currently running on the machines, in form of work in progress dates, which indicate that a machine will be busy until a given timepoint in the future. The work in progress cannot be changed by our scheduling, and must be respected in the resource allocation. It is possible that the actual end-dates of these tasks will vary, which may result in a need to reschedule a solution when new information becomes available.

When we have created a new schedule with the currently visible jobs, we only commit to the execution of an initial part of the schedule, depending on a commit horizon parameter. We may, for example, schedule all jobs that become available within the next 12 hours, but then only commit the schedule for all jobs starting in the next four hours. The idea behind this is that on one side we need to prepare the work for tasks in the near future, and should therefore commit to some work that may not start immediately. On the other hand, release dates may change, the work in progress may require more time, or some machines may become unavailable, so it is inadvisable to commit the complete future schedule.

2.2 Long Horizon

While our decision making is focussed on the short horizon problem, the overall objectives of the problem are defined over a longer time period. While we can try to minimize the waiting time for tasks in our short horizon problem, we do not know how our short-term decisions may affect the overall waiting time for a longer horizon of, say, one month. The same problem holds for running resources without long breaks, the decision made in the short term depend on decisions made before, and affect future decisions that are not visible at the given time. If we decide to stack tasks in the short horizon, we may preclude stacking tasks differently at a later timepoint.

The overall objective is to find a good compromise between the three main cost elements:

- The waiting time of the tasks between the release date and the start of the first stage, and between tasks of consecutive stages. The total waiting time for a job influences the product quality.
- The number of stacked tasks, as this increase factory throughput, and reduces energy cost, while usually also increasing the waiting time of one of the tasks involved.
- The number of resource starts over the total horizon, as each start up requires additional energy. But reducing the number of ovens that are used at the same time may again increase waiting times of some tasks, if we delay them until a resource becomes available.

These three cost elements are represented in our model as parts of the cost function, but other key performance indicators (KPIs) can be defined that allow us to compare different solutions. These are defined in the next section.

2.3 KPIs

We distinguish two types of performance indicators:

- The short term indicators can be computed from a short-horizon solution, and are only concerned with information given in the current solution. As we implement only part of that solution, these indicators are not always good predictors of long-horizon solution

Table 1 Short Term KPIs

Name	Unit	Explanation
Ovens Used	-	The number of oven used by the tasks scheduled and the existing work in progress
Percent Stacked Tasks	Percentage (0-100)	Percent of all tasks scheduled that are stacked
Percent Jobs No Wait	Percentage (0-100)	Percentage of jobs that are scheduled without any waiting time
Job Average Wait	Minutes	Average wait time over all jobs
Job Max Wait	Minutes	Largest waiting time for any job scheduled
Tasks Fixed	-	Number of tasks fixed in the schedule
Average WIP	Minutes	Average duration of non-zero work in progress
Maximal WIP	Minutes	Maximal duration of work in progress on any machine

quality, but can be used to compare individual solutions for one specific timepoint, or evaluate the difficulty of scheduling different timepoints.

- The long term indicators can be computed from the planned schedule over the long horizon, combining all elements of the short-term solutions that we committed to in each sub-problem. These indicators can be used to compare different solution methods, parameter settings, or additional constraints in our model. In this paper, we focus on the planned schedule, and ignore potential perturbations introduced by delays or break-downs.

The short term indicators are listed in Table 1, the long term indicators are listed in Table 2

3 Short Horizon Model

In this section we describe the short-term model, presenting two competing models. The machine choice model uses a traditional scheduling constraint, and does not allow task stacking. It provides a baseline result for the second model, which allows task stacking. We first start with common notations and the input data that is used.

3.1 Data and Notation

We consider a set N of $n = |N|$ jobs, and use index i to refer to specific jobs. We consider the set Q of $q = |Q|$ stages for each jobs. The stages must be executed in the given order from 1 to q, we use j as an index for a stage. We use the set M of $m = |M|$ machines, and use the index k for machines. All machines are considered to be identical.

We use the following data

r_i	release date of job i, the first task of the job cannot start before that time
d_{ij}	duration of the task in stage j of job i, does not depend on the resource allocated
t_{ij}	type of the process used for task in stage j of job i. Only tasks of the same type can be stacked. Tasks of different stages always have different types.

Table 2 Long Term KPIs

Name	Unit	Explanation
Global Time	Seconds	Total time for solving all sub problems
Nr Jobs	-	Total number of jobs scheduled
Nr Tasks	-	Total number of tasks scheduled
Percent Optimal	Percentage (0-100)	How many sub problems were solved to optimality
Percent Stacked Tasks	Percentage (0-100)	Percentage of all tasks scheduled that were stacked
Percent Jobs No Wait	Percentage (0-100)	Percentage of jobs that were scheduled without any waiting time
Job Average Wait	Minutes	Average wait time over all jobs
Job Maximal Wait	Minutes	Largest waiting time for any job scheduled
Ovens Used	-	Total number of ovens used during period
Avg Task Duration	Minutes	Average tasks duration (influenced by stacking)
Oven Runs	-	Number of oven runs over total horizon
Run Overhead Percent	Percentage (0-100)	Overhead during oven runs when machine is idle
Avg Runs per Oven Used	-	Average number of oven runs per oven used

w_k work in progress on machine k, value zero indicates no work in progress

In addition, we use the following parameters to control the model:

maxOvens	maximal number of ovens to be used
maxWait	maximal waiting time allowed for each task
maxStacked	how many tasks can be stacked together, set to two in all experiments
commitHorizon	how much of the generated schedule we are committing for execution, any job starting before that horizon is frozen for execution
l_j	boolean indicator to link stages j and $j+1$ of all jobs together, the next stage is run immediately after the end of the previous stage, on the same machine
$\alpha_1, \alpha_2, \alpha_3$	three weight factors for the cost function

3.2 Variables

s_{ij}	non-negative integer start of task j of job i
$m_{ij} \in M$	integer machine on which task j of job i is run
c_i	non-negative wait cost of job i
z_{ij}	integer number of tasks stacked together with task j of job i, ranges from 0 to *maxStacked*-1
$b_{i_1 i_2 j}$	0/1 variable to indicate if tasks for jobs i_1 and i_2 with $i_1 < i_2$ in stage j are stacked
nrOvens	integer variable between 0 and *maxOvens*, counting the number of ovens used by tasks and work in progress

3.3 Constraints

We first define the temporal constraints. Each job can only start after its release date, and the tasks of a job must be executed in order of the stages.

$$\forall_{i \in N}: \quad s_{i1} \geq r_i \tag{1}$$

$$\forall_{i \in N} \forall_{j \in 1..q-1}: \quad s_{ij+1} \geq s_{ij} + d_{ij} \tag{2}$$

Each task can wait for at most *maxWait* time periods.

$$\forall_{i \in N}: \quad s_{i1} \leq r_i + \text{maxWait} \tag{3}$$

$$\forall_{i \in N} \forall_{j \in 1..q-1}: \quad s_{ij+1} \leq s_{ij} + d_{ij} + \text{maxWait} \tag{4}$$

Adding constraint on the maximal wait time can lead to the problem becoming infeasible, when there are no more additional resources available, and waiting times steadily increase over time.

The total wait of a job is defined as the total wait time of all its tasks. The first term in (5) defines the scheduled end date of the job, the second term defines the earliest end date of the job. Note that the wait time can never be negative.

$$\forall_{i \in N}: \quad c_i = s_{iq} + d_{iq} - (r_i + \sum_{j=1}^{q} d_{ij}) \tag{5}$$

We can link the tasks of consecutive stages together, if the parameter l_j is set. This simplifies the problem, and ensures that there is no additional wait time between stages j and $j+1$.

$$\forall_{i \in N} \forall_{j \in 1..q-1}: \quad l_j \Rightarrow (s_{ij+1} = s_{ij} + d_{ij} \wedge m_{ij+1} = m_{ij}) \tag{6}$$

The following constraint counts the number of ovens used by the work in progress and the scheduled tasks. The operator *++* indicates concatenation of vectors, the *nvalue* constraint is one of the common constraints of the Global Constraint Catalog.

$$\text{nvalue}(\text{nrOvens}, [m_{ij} | i \in N, j \in Q] {++} [k | k \in M \text{ s.t. } w_k > 0]) \tag{7}$$

3.3.1 Machine Choice Resource Constraint

In a traditional scheduling model, we would express the resource constraints with a *diffn* constraint. It states that the tasks defined by the start times s_{ij} and the machine choice m_{ij}, with task duration d_{ij}, are not overlapping. Two tasks are either scheduled on different machines, or are scheduled one before the other.

$$\text{diffn}([s_{ij} | i \in N, j \in Q], [m_{ij} | i \in N, j \in Q], [d_{ij} | i \in N, j \in Q], [1 | i \in N, j \in Q]) \tag{8}$$

This is a well-known constraint which is supported in a large number of constraint solvers. The disadvantage is that we cannot stack tasks with this model. Even if we switch to a third dimension, we cannot easily express the condition that two tasks can only overlap if they have the same type, and they start and end at the same time.

3.3.2 Stacked Task Resource Model

It seems quite difficult of express the stacking condition with a disjunction of existing global constraints. We therefore define the constraints as a set of disjunctions of primitive constraints. While this formulation will not work well in traditional finite constraint solvers, it matches very well solvers based on boolean variables, or mixed integer programming solvers.

The basic diffn constraint is equivalent to the set of disjunctions

$$\begin{aligned}\forall_{i_1,i_2\in N}\forall_{j_1,j_2\in Q \text{ s.t. } <i_1,j_1>\neq<i_2,j_2>}: \quad & m_{i_1j_1} \neq m_{i_2j_2} \vee \\ & s_{i_1j_1} \geq s_{i_2j_2} + d_{i_2j_2} \vee \\ & s_{i_2j_2} \geq s_{i_1j_1} + d_{i_1j_1}\end{aligned}$$

We need to add a fourth case stating that if the types of the tasks are the same, they can be scheduled on the same machine, starting and ending at the same time. The stacking can only occur if the tasks belong to the same stage j, if the stages are different, only the original three alternatives are allowed. This can be expressed by the constraint set

$$\begin{aligned}\forall_{i_1,i_2\in N}\forall_{j_1,j_2\in Q \text{ s.t. } j_1\neq j_2}: \quad & m_{i_1j_1} \neq m_{i_2j_2} \vee & (9)\\ & s_{i_1j_1} \geq s_{i_2j_2} + d_{i_2j_2} \vee & (10)\\ & s_{i_2j_2} \geq s_{i_1j_1} + d_{i_1j_1} & (11)\end{aligned}$$

which constrains all task pairs belonging to different stages.

If the stages are the same, we begin with the following constraint formulation, which adds the case where two tasks of the same type can be stacked together.

$$\begin{aligned}\forall_{i_1,i_2\in N \text{ s.t. } i_1\neq i_2}\forall_{j\in Q}: \quad & m_{i_1j} \neq m_{i_2j} \vee \\ & s_{i_1j} \geq s_{i_2j} + d_{i_2j} \vee \\ & s_{i_2j} \geq s_{i_1j} + d_{i_1j} \vee \\ (t_{i_1j_1} = t_{i_2j_2} \wedge m_{i_1j} = m_{i_2j} \wedge s_{i_1j} = s_{i_2j}) &\end{aligned}$$

This set of disjunctions allows stacking of tasks, but any number of compatible tasks can be stacked together. If we want to limit the number of tasks that can be stacked together, we need some additional variables and constraints. First we introduce 0/1 variables $b_{i_1i_2j}$ to check if two jobs i_1 and i_2 are stacked together in stage j. To avoid double counting we only define these variables for $i_1 < i_2$. In our disjunction, the b variable is equal to one only in the last case, and zero in all other cases. We can add these variables in the following set of disjunctions, which define the constraints to use in our model.

$$\begin{aligned}\forall_{i_1,i_2\in N \text{ s.t. } i_1<i_2}\forall_{j\in Q}: \quad & (b_{i_1i_2j} = 0 \wedge (m_{i_1j} \neq m_{i_2j} \vee & (12)\\ & s_{i_1j} \geq s_{i_2j} + d_{i_2j} \vee & (13)\\ & s_{i_2j} \geq s_{i_1j} + d_{i_1j}) \vee & (14)\\ (b_{i_1i_2j} = 1 \wedge t_{i_1j_1} = t_{i_2j_2} \wedge m_{i_1j} = m_{i_2j} \wedge s_{i_1j} = s_{i_2j}) & & (15)\end{aligned}$$

The variable z_{ij} counts how many other tasks task j of job i is stacked together with. We know that any stacked tasks must be from the same stage, as the types of the tasks in other stages cannot be equal to t_{ij}. We sum over all relevant b variables, keeping in mind that $b_{i_1i_2j}$ is only defined for $i_1 < i_{i_2}$.

$$\forall_{i\in N}\forall_{j\in Q}: \quad z_{ij} = \sum_{i_1=1}^{i-1} b_{i_1ij} + \sum_{i2=i+1}^{n} b_{ii_2j} \qquad (16)$$

There is a global limit on how many tasks can be stacked together, the z variables must be below that limit *maxStacked*, taking the task itself into account.

$$\forall_{i \in N} \forall_{j \in Q}: \quad z_{ij} < \text{maxStacked} \tag{17}$$

We also need to express that tasks cannot overlap the work in progress. This leads to another set of disjunctions, stating that any task either starts after the end of the work in progress on machine k, or is not scheduled on that machine.

$$\forall_{k \in M} \forall_{i \in N} \forall_{j \in Q}: \quad (w_k \leq s_{ij} \vee m_{ij} \neq k) \tag{18}$$

A large number of these constraints can be eliminated by preprocessing if the release date r_i of job i is after the end of the work in progress on machine k.

3.4 Objective

The objective is a weighted sum of the three main cost elements, the waiting time, the number of stacked tasks, and the number of ovens used. We use weight factors α to choose between different solutions. In our experiments, we set $\alpha_1 = 1$, and scale the other two factors accordingly. A value of 1000 for α_2 for example means that we equate using an extra oven to an increase of total waiting time of 1000 minutes.

$$\min \alpha_1 \sum_{i \in N} w_i + \alpha_2 \text{nrOvens} + \alpha_3 \sum_{i \in N, j \in Q} z_{ij} \tag{19}$$

We minimize the objective, we therefore use a negative value for α_3 as we are interested in creasing the number of stacked tasks.

This objective (19) together with constraints (1)-(18) forms our constraint model for the short horizon.

3.5 Committing Solution

The constraint model defined above finds a solution considering all visible jobs, but we only want to commit to the initial part of the schedule. In an interactive tool, we can allow the user to look at the schedule to decide which part of the schedule should be committed, but in order to run experiments without constant user interaction, we need to select committed jobs automatically. At the minimum, we need to commit jobs that are scheduled to start at the current point in time. For practical reasons, we may want to extend the commit horizon into the future in order to allow task preparation. For our experiments, we use the *commitHorizon* parameter, and select all jobs that start within that horizon, i.e. jobs satisfying the condition

$$s_{i1} \leq \text{commitHorizon} \tag{20}$$

In our industrial use case, we fix the job completely for all stages, in a more general setting, we can allow to reschedule later stages of the job in the next iteration.

4 Solving Individual Instances

The data for the experiments were provided by one of the industrial partners in the ASSIST-ANT project, we extended the dataset to allow for a more comprehensive evaluation, details are provided in Appendix A.

We first present some example results from individual sub problems. This is the type of information that would be available in an operational use of the tool, solving one sub problem at a time, and giving the user the information to decide whether to implement the solution suggested.

Figure 1 shows a Gantt chart of the different jobs included in a sub problem, the fixed Kuka tasks are shown in red, they define the release dates for jobs of our problem. There are two stages (shown in green and blue), with stacked tasks shown in a darker colour. Tasks waiting are indicated in red, the selected jobs for this solution are outlined in blue. The commit and lookahead horizons are shown by vertical lines, four and 12 hours into the future. We see that the solution suggests to stack three pairs of tasks with the same product type, given by the label inside the task. In each stacking case, one of the tasks has to wait until the second task becomes available. Two other, non-stacked tasks are waiting for a currently active machine to become available, instead of using an additional, currently unused resource.

Figure 1 Job Gantt Chart showing release dates, scheduled tasks for two stages, indicating waiting times for tasks, and highlighting stacked tasks with darker color

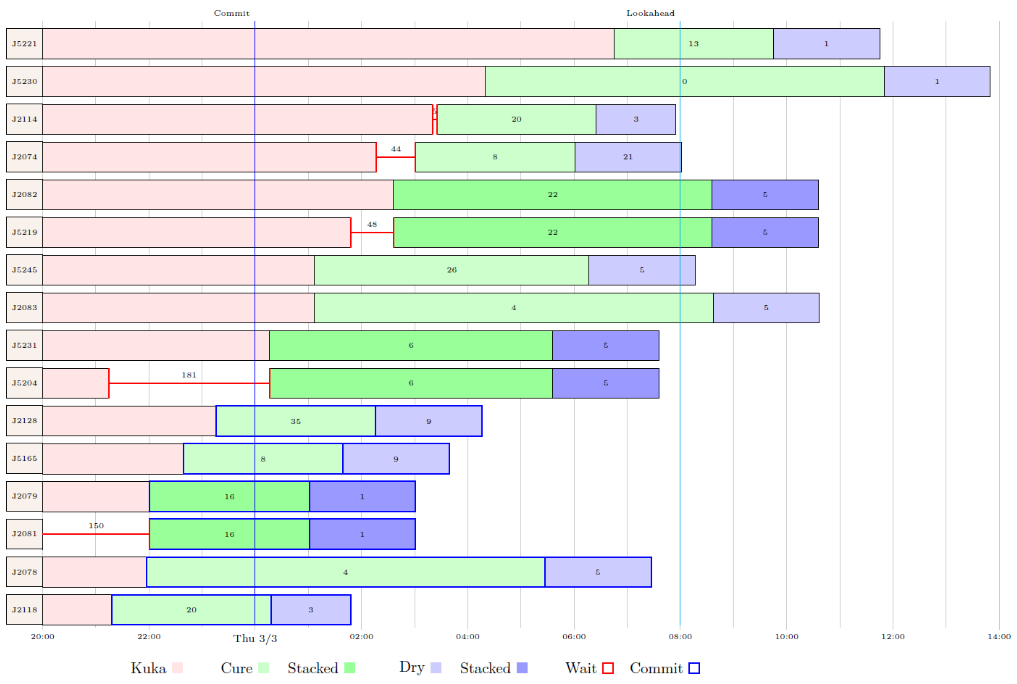
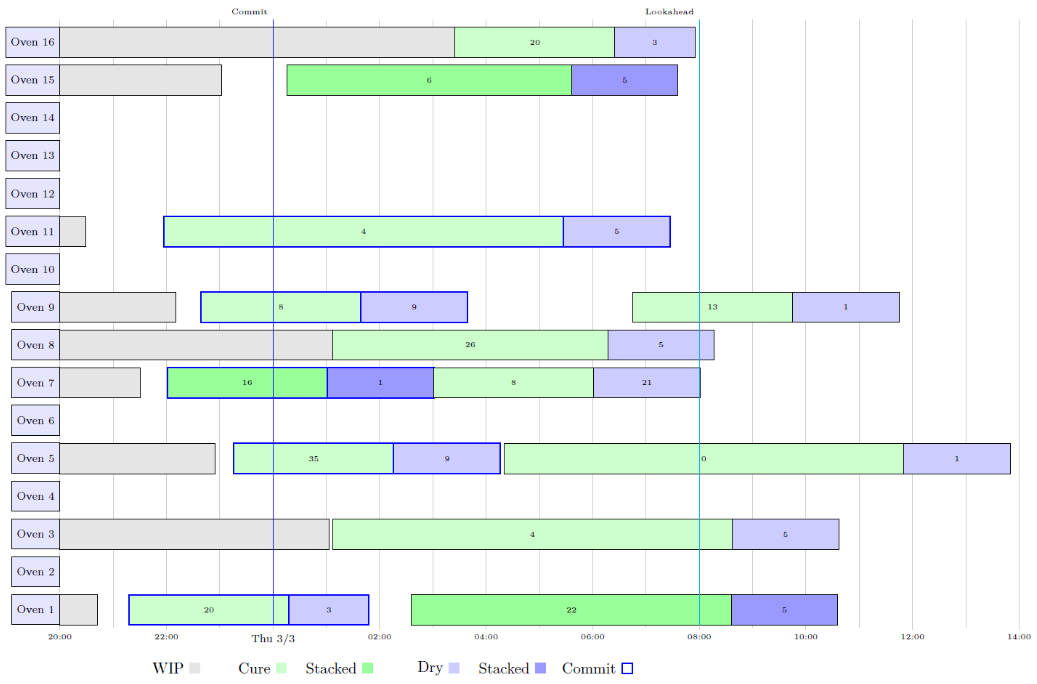


Figure 2 shows the resource view for the given solution. Fixed work in progress originating in earlier sub problems are shown in gray, the tasks of the two stages are again shown in green and blue. In the solution we use 9 of the available 16 ovens, there is only one gap leading to a shut-down and following restart for oven 9. The tasks outlined in blue are frozen, and will become work in progress for the next sub problem.

For each run of the solver, we compute a set of short-term KPIs, as shown in Figure 3. We plot the KPIs as line plots, giving an indication how the values are changing over consecutive

Figure 2 Resource Gantt Chart, showing work in progress in gray, linked tasks for two stages, using five of 16 available ovens, jobs to be commited are shown with a blue outline



sub problems. We see for example that the number of tasks peaked at 42 four time periods before, while both cost and number of ovens used have reached a plateau in the last runs. We can also use this type of view to compare different option settings for the same time period, where the user might experiments with different choices to explore more alternative solutions.

5 Long Horizon Solving

By default, we run each experiment for 200 timepoints with a 4 hour horizon. This creates a schedule of just over one month duration. Figure 4 shows the resulting overall schedule on the resources. We can clearly identify periods of lower and higher demand, resulting in the use of more or fewer resources. The scheduler does not try to schedule jobs always on the same resources, there is no constraint enforcing such a condition. Figure 5 shows the resulting long-term KPIs for the same schedule.

To make the oven runs, the periods where the ovens are active over a longer period of time, more visible, we also present a Gantt chart abstracting the tasks into oven run activities, shown in Figure 6. Note that our model does not explicitly minimize the oven runs, we control a proxy measure, the number of ovens used in each period instead. There may be future improvements where we model the oven runs explicitly, but as they may stretch over multiple sub problems, this is not obvious.

6 Global Comparison

We now present results for a number of experiments that we ran to investigate the impact of different design choices and parameter values.

Figure 3 Sub Problem KPI Comparison

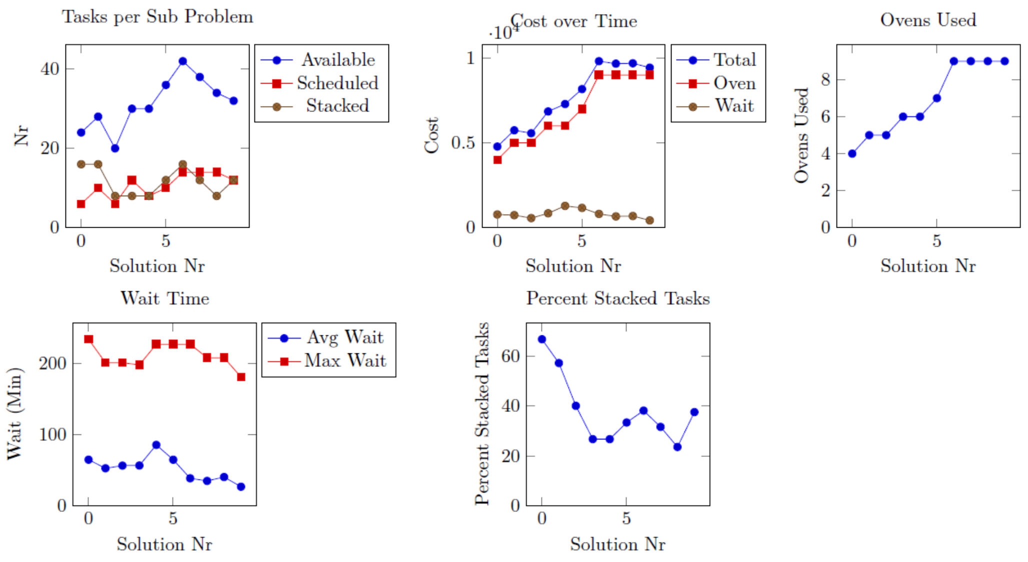


Figure 4 Generated Oven Schedule for 200 runs with commitHorizon of 4 hours

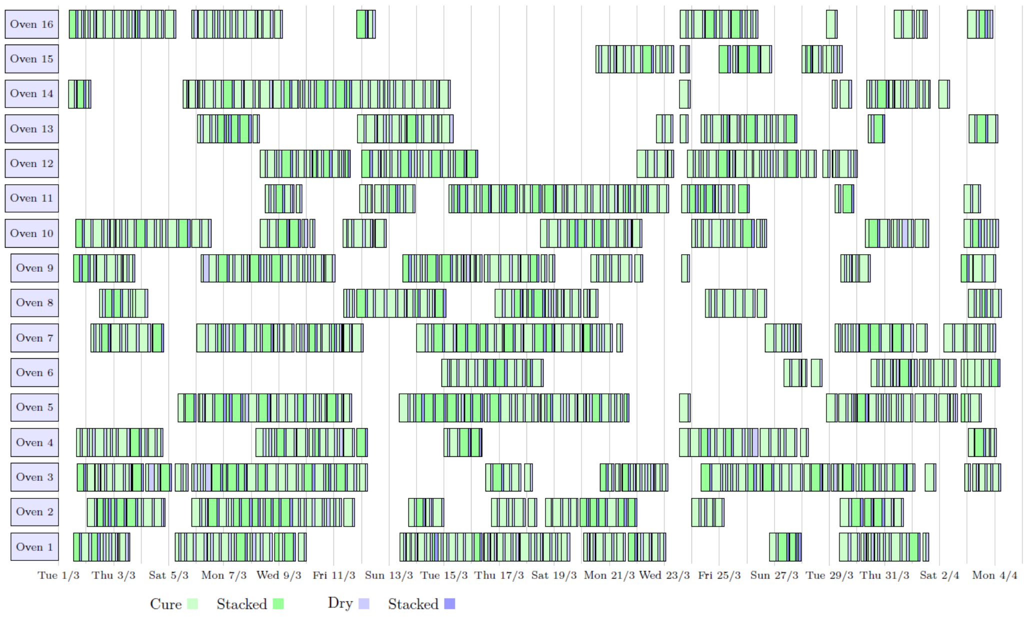
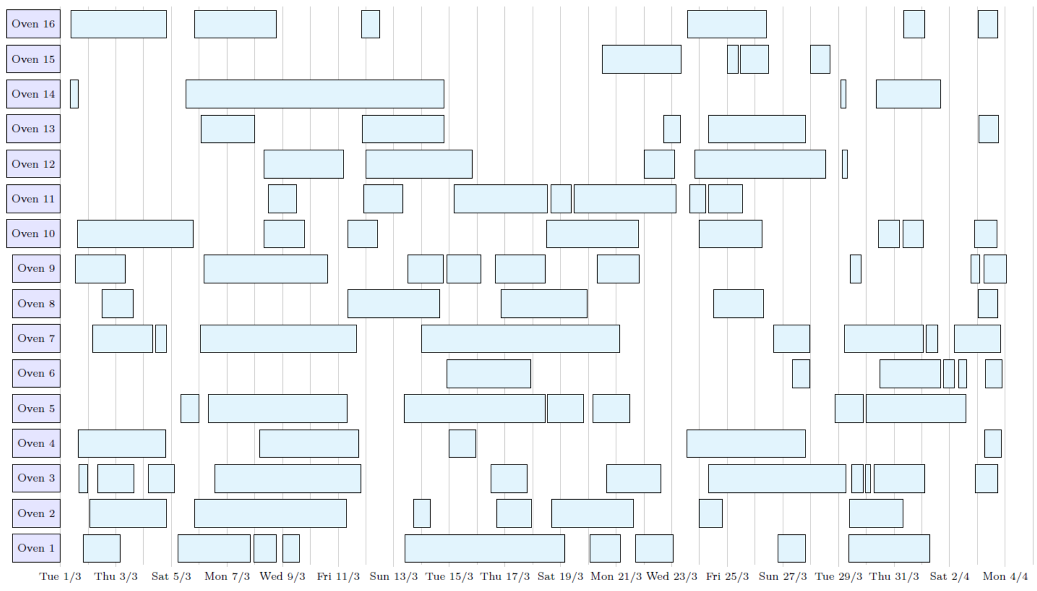


Figure 5 Long Horizon KPIs for Schedule in Figure 4

Global Time (s)	Nr Jobs	Nr Tasks	Percent Optimal	Stacked Tasks	Percent Stacked Tasks	Total Wait (min)	Jobs No Wait	Percent Jobs No Wait	Job Avg Wait	Job Max Wait	Ovens Used	Total Task Duration	Avg Task Duration	Oven Runs	Min Run Duration	Avg Run Duration	Max Run Duration	Run Overhead Percent	Avg Runs per Oven Used
776.06	891	1,782	71.50	688	38.61	60,640	283	31.76	68.06	240	16	298,575	167.55	106	260	2,883.38	13,383	2.37	6.63

Figure 6 Extracted Oven Runs for Schedule in Figure 4



We consider the following research questions:

- How do different open-source and commercial solvers compare on problems of increasing size? Is the approach scalable for realistic datasets?
- How do the two alternative resource models compare to each other? Is the increased complexity of the stacked task model justified by the results obtained?
- How do choices for the parameters *commitHorizon*, *maxWait*, and *lookahead* affect the results?
- What are good values for the α weight factors in the objective function to find a good compromise between the conflicting objective function terms?
- We limit the amount of time spent in each sub problem by a timeout limit. How does that limit affect overall solution quality?
- In the data preparation, we diversify the product types of copied jobs to create a more complex problem. How does the percentage of modified products affect the complexity of the problem?

6.1 Comparing Different Back-end Solvers and Problem Sizes

In the first experiment we run the solver for 200 timepoints on datasets of increasing size. For this we create multiple (from 0 to 19) copies of the original jobs, varying both duration and release date by random offsets, and changing the product type for 70% of the copied tasks, to create a more varied order set.

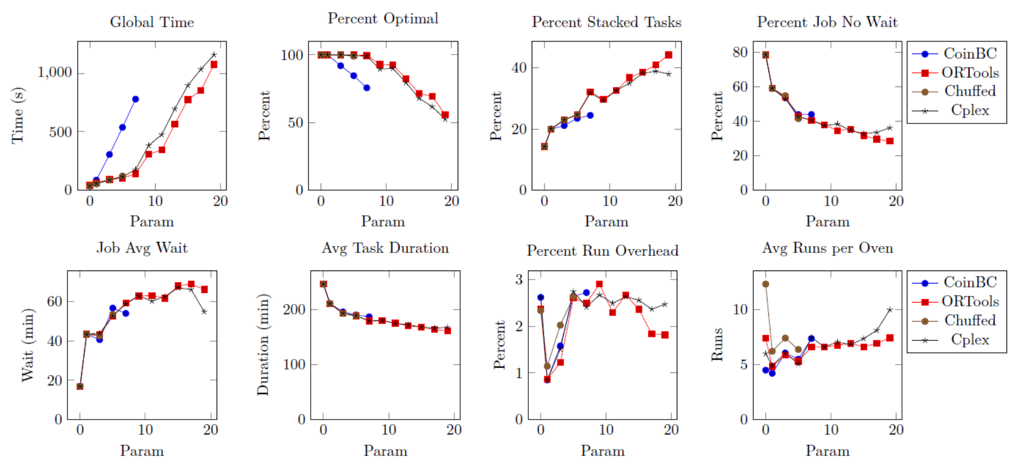
We tried four different back-end solvers for our Stacked model:

Chuffed is a SAT based finite domain solver, developed by the MiniZinc team, and packaged with MiniZinc. The solver can use only a single worker thread.

Coin-BC is an open-source MIP solver packaged with MiniZinc. We allow to use up to 25 worker threads for the experiments.

OR-Tools is a constraint solver developed by Google, it comes packaged with an interface for MiniZinc. This solver does not perform well when limited to a single thread, we allow

Figure 7 Impact of Solver Type on KPIs with Increasing Problem Size



to use up to 25 worker threads.

Cplex is a commercial MIP solver offered by IBM, which can be integrated into MiniZinc. We also allow it to use up to 25 worker threads.

In Figure 7, we show the results obtained for different global KPIs. Chuffed can only solve the smaller problem instances, and fails to find a solution within the given time limit as the problem size increases. As soon as one sub problem cannot be solved, we abort the run, as there is little chance of continuing on the time-line.

Coin-BC works slightly better, but cannot keep up with the two successful solvers, OR-Tools and Cplex. Both find good, but not always optimal, solutions within the given time limit (by default 10 seconds for each sub problem).

For the following experiments, we always use OR-Tools. All experiments were run on a Windows 11 desktop, with a Intel(R) Xeon(R) W-2275 CPU @ 3.30GHz, and 128GB of memory. The CPU has 14 cores, we allow the solvers to use up to 25 threads.

6.2 Impact of *commitHorizon* Parameter

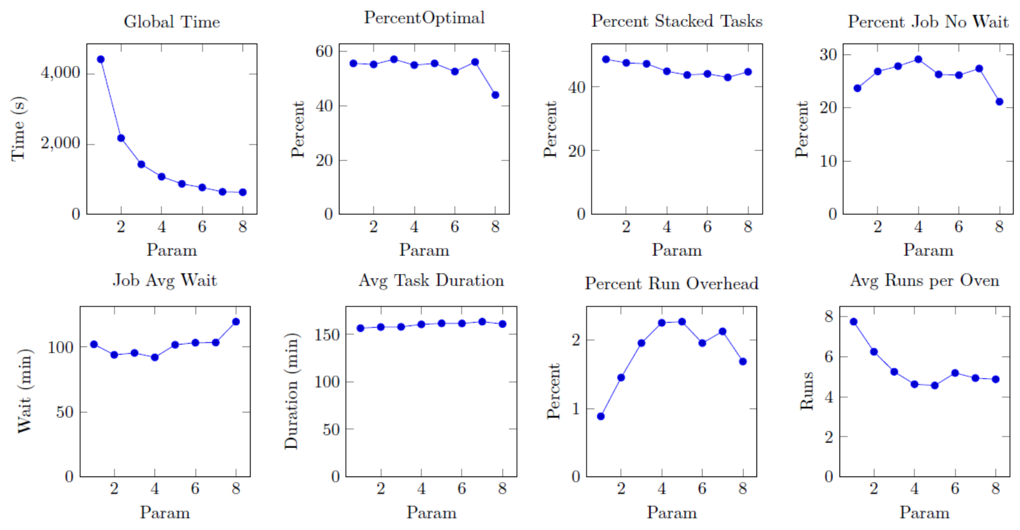
For the next experiment we vary the *commitHorizon* parameter between one and eight hours, setting the lookahead to eight hours for all runs. As the horizon shrinks, we need to run more sub problems to cover the same, one month, overall planning horizon. As we can see in Figure 8, the shorter horizon therefore requires much more time to run. Surprisingly, most of the KPIs do not seem to be affected by the horizon, except for the average number of oven runs, which decreases with increasing horizon.

Note that in this experiment we did not introduce any variation of the release dates or task duration, which we might expect in a real-world scenario.

7 Conclusion

In this paper we have looked at an oven scheduling problem arising in the manufacturing of compressor rotors to illustrate the importance of short and long horizon models to optimize the overall manufacturing process. Choices taken in the short-term model affect future runs of the model, while only limited data about future orders is available in the short term. The

Figure 8 Impact of *commitHorizon* Parameter



choice of different parameters controls how well the short-term solutions help to find an overall solution of high quality. Unusual for CP based scheduling models, our model avoids the use global constraints in favour of quite simple disjunctions of primitive constraints, which make the model well suited for CP and SAT based solvers, as well as different MIP solvers. The paper illustrates how a proxy objective in the short horizon model can help to find good solutions over a longer time horizon.

References

1 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993. URL: https://www.sciencedirect.com/science/article/pii/089571779390068A, doi:10.1016/0895-7177(93)90068-A.

2 Philippe Baptiste. Constraint-based schedulers, do they really work? In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, page 1. Springer, 2009. doi:10.1007/978-3-642-04244-7_1.

3 Philippe Baptiste, Philippe Laborie, Claude Le Pape, and Wim Nuijten. Constraint-based scheduling and planning. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 761–799. Elsevier, 2006. doi:10.1016/S1574-6526(06)80026-X.

4 Roman Barták and Miguel A. Salido. Constraint satisfaction for planning and scheduling problems. *Constraints An Int. J.*, 16(3):223–227, 2011. URL: https://doi.org/10.1007/s10601-011-9109-4, doi:10.1007/S10601-011-9109-4.

5 Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, Malgorzata Sterna, and Jan Weglarz. Constraint Programming and Disjunctive Scheduling. In *Handbook on Scheduling*, International Handbooks on Information Systems, chapter 16, pages 609–670. Springer, November 2019. URL: https://ideas.repec.org/h/spr/ihichp/978-3-319-99849-7_16.html, doi:10.1007/978-3-319-99849-7.

6 Silvia Breitinger and Hendrik C. R. Lock. Using constraint logic programming for industrial scheduling problems. In Christoph Beierle and Lutz Plümer, editors, *Logic Programming:*

Formal Methods and Practical Applications, Studies in Computer Science and Artificial Intelligence, pages 273–299. Elsevier Science B.V./North-Holland, 1995.

7 Giacomo Da Col and Erich C. Teppan. Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9:100249, 2022. URL: `http://dx.doi.org/10.1016/j.orp.2022.100249`, `doi:10.1016/j.orp.2022.100249`.

8 Cyrille Dejemeppe. *Constraint programming algorithms and models for scheduling applications*. PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2016. URL: `https://hdl.handle.net/2078.1/178078`.

9 Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving large combinatorial problems in logic programming. *The Journal of Logic Programming*, 8(1):75–93, 1990. `doi:10.1016/0743-1066(90)90052-7`.

10 Mark S. Fox, Bradley P. Allen, and Gary Strohm. Job-shop scheduling: An investigation in constraint-directed reasoning. In David L. Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, USA, August 18-20, 1982*, pages 155–158. AAAI Press, 1982. URL: `http://www.aaai.org/Library/AAAI/1982/aaai82-037.php`.

11 John N. Hooker. A hybrid method for the planning and scheduling. *Constraints An Int. J.*, 10(4):385–401, 2005. URL: `https://doi.org/10.1007/s10601-005-2812-2`, `doi:10.1007/S10601-005-2812-2`.

12 John J. Kanet, Sanjay Ahire, and Michael F. Gorman. Constraint programming for scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. URL: `http://www.crcnetbase.com/doi/abs/10.1201/9780203489802.ch47`, `doi:10.1201/9780203489802.CH47`.

13 Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, October 2006. URL: `http://dx.doi.org/10.1016/j.ejor.2005.01.065`, `doi:10.1016/j.ejor.2005.01.065`.

14 Rainer Kolisch and Arno Sprecher. Psplib - a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, January 1997. URL: `http://dx.doi.org/10.1016/s0377-2217(96)00170-1`, `doi:10.1016/s0377-2217(96)00170-1`.

15 Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG. *Constraints An Int. J.*, 23(2):210–250, 2018. URL: `https://doi.org/10.1007/s10601-018-9281-x`, `doi:10.1007/S10601-018-9281-X`.

16 Marie-Louise Lackner, Christoph Mrkvicka, Nysret Musliu, Daniel Walkiewicz, and Felix Winter. Minimizing cumulative batch processing time for an industrial oven scheduling problem. In Laurent Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: `https://doi.org/10.4230/LIPIcs.CP.2021.37`, `doi:10.4230/LIPICS.CP.2021.37`.

17 Marie-Louise Lackner, Christoph Mrkvicka, Nysret Musliu, Daniel Walkiewicz, and Felix Winter. Exact methods for the oven scheduling problem. *Constraints An Int. J.*, 28(2):320–361, 2023. URL: `https://doi.org/10.1007/s10601-023-09347-2`, `doi:10.1007/S10601-023-09347-2`.

18 Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints An Int. J.*, 17(1):51–85, 2012. URL: `https://doi.org/10.1007/s10601-011-9115-6`, `doi:10.1007/S10601-011-9115-6`.

19 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume

4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. `doi:10.1007/978-3-540-74970-7_38`.

20 Laurent Perron, Frédéric Didier, and Steven Gay. The cp-sat-lp solver. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:2, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2023/19040`, `doi:10.4230/LIPIcs.CP.2023.3`.

21 Jean-Francois Puget. Applications of constraint programming. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming - CP'95, First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings*, volume 976 of *Lecture Notes in Computer Science*, pages 647–650. Springer, 1995. `doi:10.1007/3-540-60299-2_43`.

22 Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. *A Satisfiability Solving Approach*, pages 135–160. Springer International Publishing, Cham, 2015. `doi:10.1007/978-3-319-05443-8_7`.

23 Helmut Simonis. The CHIP system and its applications. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming - CP'95, First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings*, volume 976 of *Lecture Notes in Computer Science*, pages 643–646. Springer, 1995. `doi:10.1007/3-540-60299-2_42`.

24 Helmut Simonis. Models for global constraint applications. *Constraints An Int. J.*, 12(1):63–92, 2007. URL: `https://doi.org/10.1007/s10601-006-9011-7`, `doi:10.1007/S10601-006-9011-7`.

25 Touraïvane. Constraint programming and industrial applications. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming - CP'95, First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings*, volume 976 of *Lecture Notes in Computer Science*, pages 640–642. Springer, 1995. `doi:10.1007/3-540-60299-2_41`.

26 Mark G. Wallace. Practical applications of constraint programming. *Constraints An Int. J.*, 1(1/2):139–168, 1996. `doi:10.1007/BF00143881`.

27 Mohammad Hossein Fazel Zarandi, Ali Akbar Sadat Asl, Shahabeddin Sotudian, and Oscar Castillo. A state of the art review of intelligent scheduling. *Artif. Intell. Rev.*, 53(1):501–593, 2020. URL: `https://doi.org/10.1007/s10462-018-9667-6`, `doi:10.1007/S10462-018-9667-6`.

A Data Sources

In this section we describe the datasets used, describe modifications to the data which allow us to perform a wider range of experiments, and then show some results which highlight the information available to the user, as well as some quantitative evaluation of the impact of solver choice, model alternatives, and parameter settings.

An initial dataset was provided by Atlas Copco, which covers the oven jobs coming from a subset of Kuka robot cells in the factory, together with the process information about task duration and temperature profile required by each product type. The data covers most of 2022, but contains only a part of all oven tasks. Early on it was decided not to include the oven tasks originating in manual work cells, as no automated data feed for this data was available.

In order to make the problem size more realistic, and to allow for some scalability experiments to test solver performance, we decided to create additional jobs from the given data set. We add *multiplier* (0-19) copies of the original jobs, and perform the following modifications:

- The start date of the job is modified by adding a uniform random value between 0 and 2800 minutes (2 days), creating a random offset of the start date. This preserves most of the time distribution of the jobs over the planning period.
- The duration of the Kuka task for the job is modified by adding a uniform random value between 0 and 120 minutes (2 hours). This increases the number of jobs visible for the oven scheduler, increasing problem complexity.
- For a specified percentage of the copied jobs (by default 70%), a new product type is randomly selected. This is intended to increase the number of different products used in each time period, making it more difficult to stack tasks. This is justified by the limited variety of products handled by the robot cells for which data was provided. We performed some experiment (shown in Section C.5) varying this percentage, to check the overall impact.

B Default Parameter Choices

For the experiments, we use the set of default parameter values shown in Table 3, unless overridden in the experiment.

C Additional Scenarios

In this appendix, we present results for some additional scenarios that might be of interest to the reader.

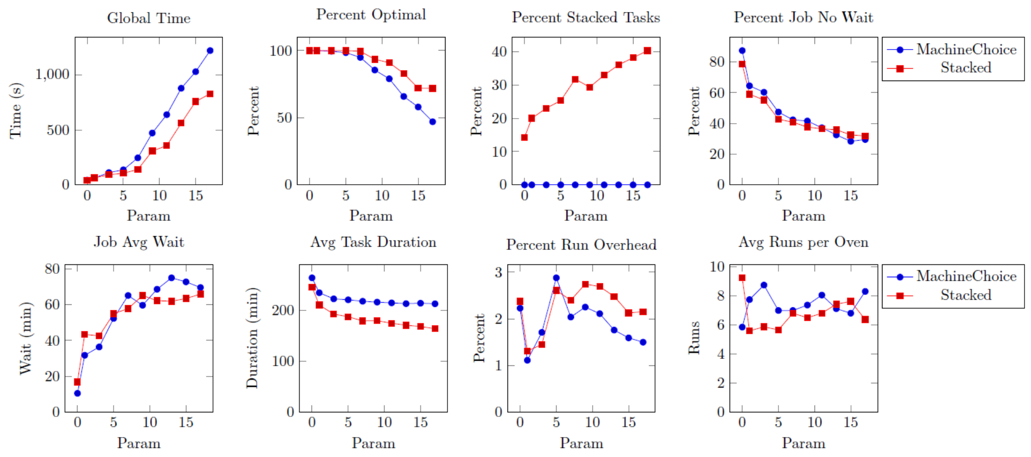
C.1 Impact of Resource Model

In this experiment we compare the two resource models, the MachineChoice model introduced in Section 3.3.1, and the StackedTasks model from Section 3.3.2. Surprisingly, the StackedTask model is faster to solve than the MachineChoice model, and obtains a higher percentage of optimal solutions. This points to limitations in the implementation of the *diffn* constraint in the OR-Tools solver. In terms of solution quality, the main difference is the percentage of stacked stacks which increases with larger problem sizes for the StackedTask model, which also leads to shorter overall average task duration. Most other KPIs, like percentage of jobs without wait, are comparable between the different approaches.

Table 3 Default Parameter Choices

Parameter	Unit	Value
nrSteps	-	200
resourceType	-	Stacked
multiplier	-	15
maxWait	Hours	4
lookahead	Hours	12
maxOvens	-	16
maxStacked	-	2
link stages	-	true
commitHorizon	Hours	4
weightOven	-	1000
weightStacked	-	0
weightWait	-	1
solverType	-	ORTools
nrThreads	-	25
Timeout	Seconds	10

Figure 9 Comparing Stacked and Machine Choice Models



C.2 Impact of Weight Factors

As our objective function is a weighted sum of three incomparable cost terms, we need to study the pact of different weight factors on the solution. In this set of experiments, we keep the weight on the job wait time $\alpha_1 = 1$ constant, and increase the wait associated with the number of ovens, or decrease the weight on stacked tasks. Results for these experiments are shown in Figures 10 and 11. The oven weight has a clear impact on the solution quality, a value of 0 leads to no stacking, and a very high number of oven runs.

Figure 10 Impact of Oven Weight Factor α_2

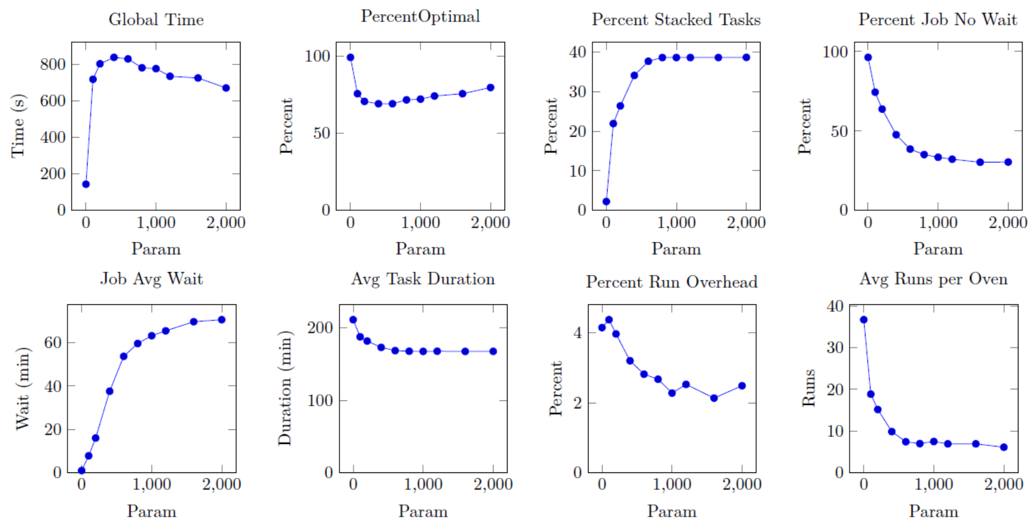
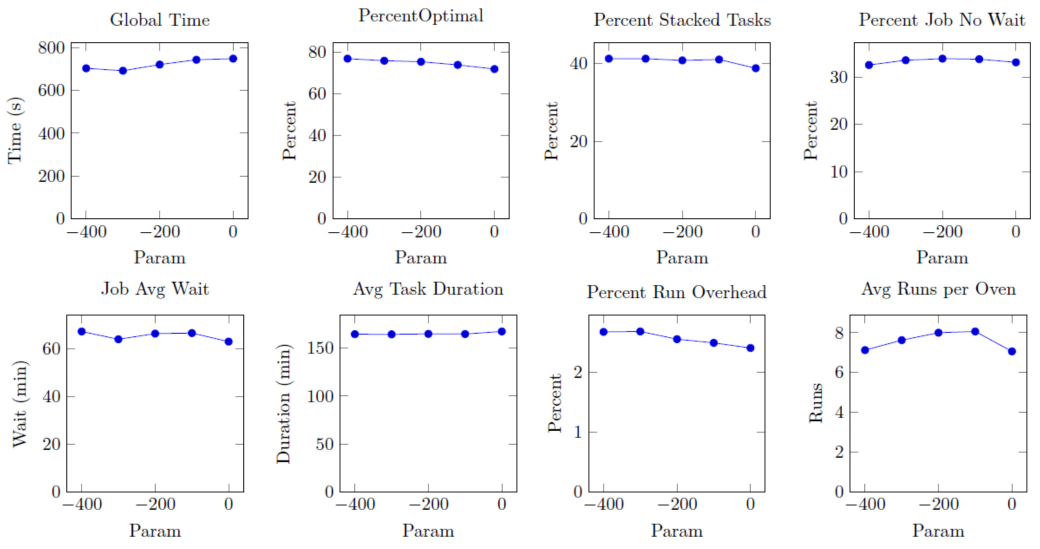


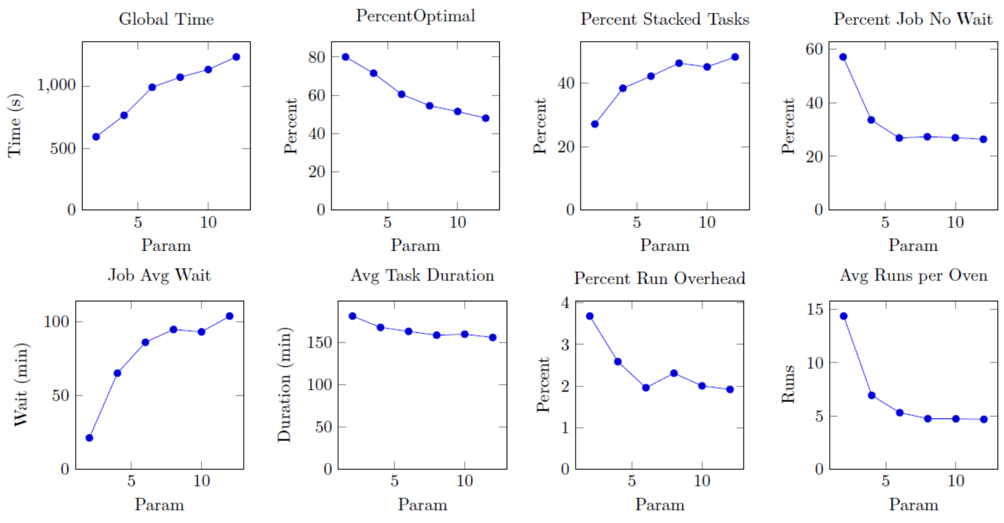
Figure 11 Impact of Stacked Weight Factor α_3



C.3 Impact of *maxWait* Parameter

For the next experiment, we vary the *maxWait* parameter between 2 and 12 hours. As we allow for more waiting time, there is an increased possibility of stacking tasks of the same type with different release dates. This increases problem complexity, as seen in Figure 12 by an increase in time needed and percent of optimal solutions found. On the other hand, the percentage of stacked tasks increases, and the average duration of the tasks decreases. this improvement comes at the price of average wait time, which increases with increasing *maxWait*.

Figure 12 Impact of *maxWait* Parameter



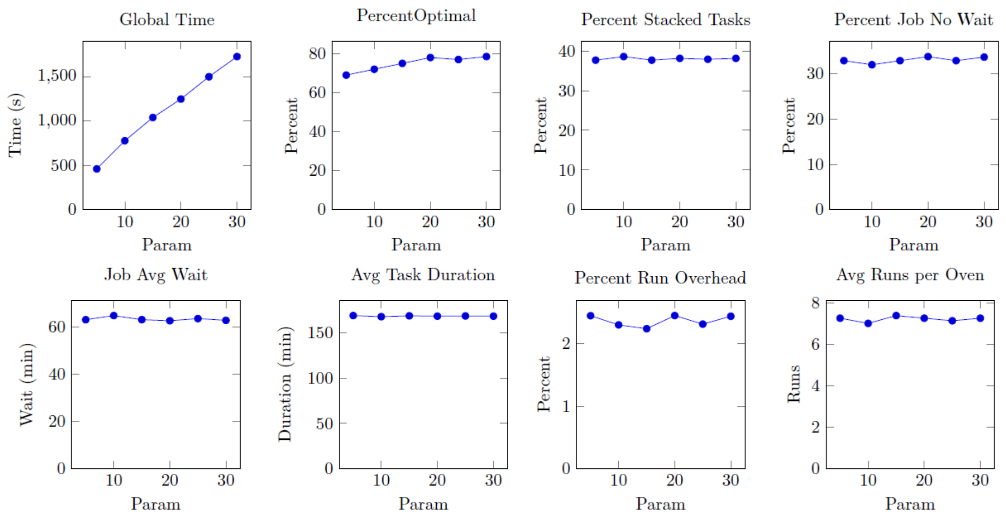
C.4 Impact of Timeout

For completeness, we show the impact of the timeout value on results in Figure 13, where we vary the timeout limit for each subproblem between five and 30 seconds. Increasing the timeout unsurprisingly leads to a longer over time to solve the problem, and a slight improvement of the percentage of sub-problems solved to optimality. But the other KPIs do not seem to be affected, leading to a conclusion that a short time limit is sufficient to obtain good overall results. On the other hand, the limit should be sufficiently large so that a feasible solution is found for each subproblem.

C.5 Impact of Random Product Selection

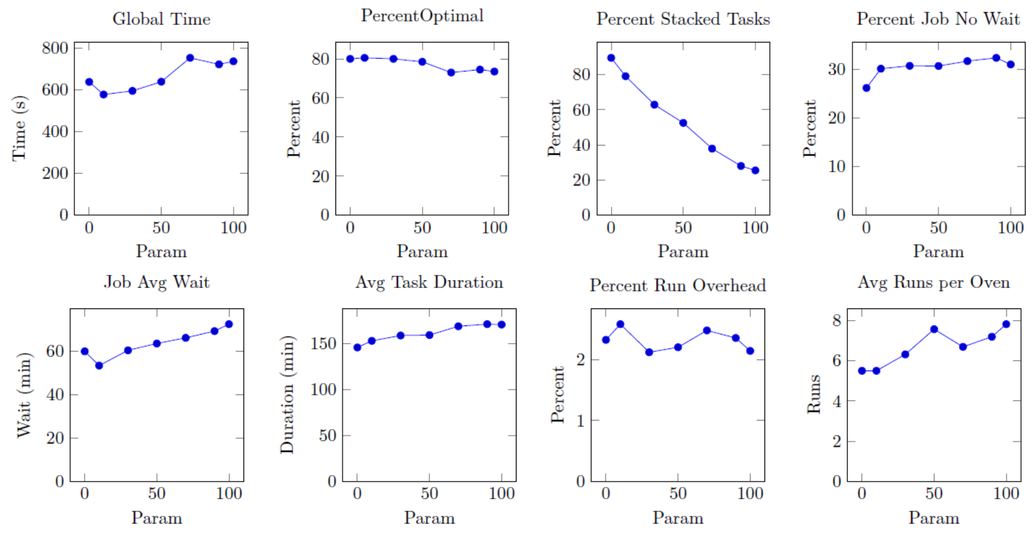
In the data preparation phase, we create copies of the original oven jobs to study scalability of the solver. By default we assign 70% of the generated jobs to a random product type. This experiment studies the impact of this percentage parameter, reassigning 0, 10, 30, 50, 70, 90, and 100% of the copied jobs. Figure 14 shows the result on the global KPI values. As we reassign more and more jobs to a random product, the chance of stacking decreases, while the average waiting time increases, probably because we have to wait longer for a second job with the same product type. The number of runs per oven increases, but not uniformly, and not be a large amount. the value of 70% seems to indicate the most complex scenario,

Figure 13 Impact of Timeout Limit on Sub-Problem solution



where there is still possibility for large amount of stacking, without increase required time too much.

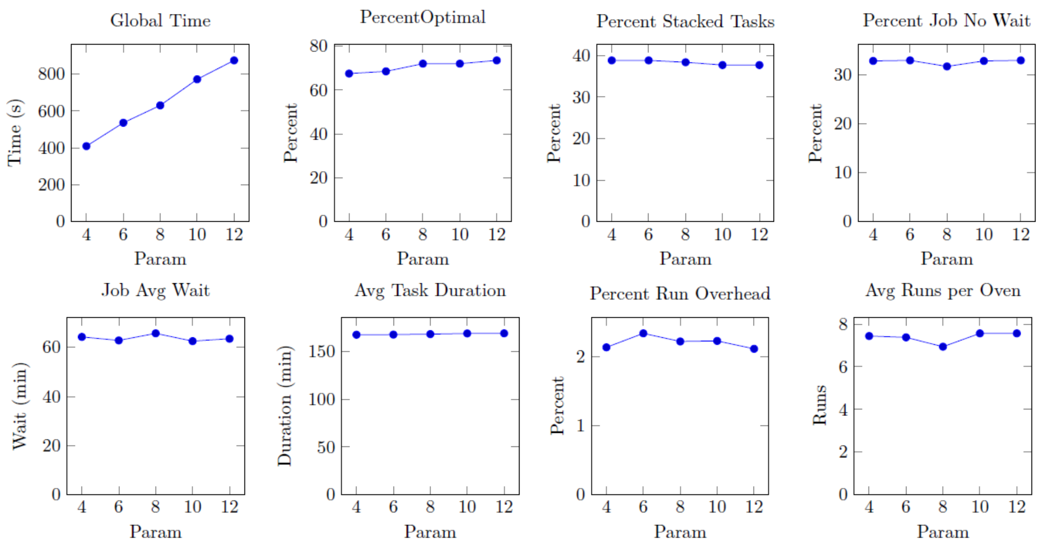
Figure 14 Impact of Random Product Selection in the Data Preparation



C.6 Impact of *lookahead* Parameter

We do not always see a dramatic change in the KPIs when we modify a problem parameter. Figure 15 shows the impact of the lookahead horizon. When we increase the horizon, we increase the size of the sub-problems to be solved, which results in an increase of the time needed to solve the problem. All other KPI values are influenced only slightly by this choice, indicating that a small value (but equal or larger than the *commitHorizon* may be sufficient to solve the problem to a high quality.

Figure 15 Impact of *lookahead* parameter


Global Time
Time (s)
Param
PercentOptimal
Percent
Percent Stacked Tasks
Percent Job No Wait
Job Avg Wait
Wait (min)
Avg Task Duration
Duration (min)
Percent Run Overhead
Avg Runs per Oven
Runs