Efficient Batch Assignment for Parallel-Machine Production Scheduling

Christoph Fünfzig[1], Roderich Wallrath[2] and Stefan Hubert[2]
[1]*CFS Software, 53894 Mechernich, Germany*
[2]*Engineering & Technology, Bayer AG, 51368 Leverkusen, Germany*

Keywords: Constraint Programming, Symmetry Breaking, Assignment Problem, Parallel-Machine Production Scheduling.

Abstract: In this article, we consider different batch assignment schemes for non-separable, non-preemptive production scheduling on m parallel work stations. Batch assignment is a very important part of production models as batches with identical parameters usually occur in a large number in real-world applications, which causes a large number of symmetric solutions. The common scheme **Exactly-n** for assignment of n batches to m machines originating from general assignment problems is very inefficient when it comes to scheduling, even with additional ordering for symmetry breaking (**Exactly-n Ordered**). We define three restricted assignment schemes for the same-parameter batches by forming blocks of size $0 \le z_i \le n$ on machine i and ordering them between the different machines. We compute bounds for the number of feasible assignments as a measure for the feasible space and give solving times from our experiments with a boolean inference-based solver like the Google CP-SAT solver. We show that with the proposed restricted assignment schemes, production scheduling models result that solve significantly faster than the models with the common scheme **Exactly-n**.

1 INTRODUCTION

In resource constraint production scheduling problems of chemical and pharmaceutical manufacturing processes, raw materials go through processing stages as batches to satisfy a given demand of product types u. The number and type of batches and work stations (machines in scheduling terminology) can be depicted as a bipartite graph (V,E), see Figure 1. In particular, the nodes V represent batches (first column) or work stations (second column) while the edges in E represent the possible material transfer routes and result in different assignment possibilities. Work stations in the first production stage process batches of the several product types and have different parameters like volume, processing rate per product type and possibly waiting and changeover times. Among the batches of same product type, most have a given same volume suitable for work stations. Some have a remainder volume to achieve a given demand.

The problem of batch process scheduling consists of two parts, assignment of each of n batches of same product type to one of m machines with possibly different parameters and disjunctive scheduling of assigned batches on each machine i ($i = 1,\ldots,m$) based on batch and machine parameters, resulting in the batch processing time on this machine. The solving time of the production scheduling model depends in a complex way on both model parts. In the following, we consider different schemes for assignment of n batches $b_{u,j}$ ($j = 1,\ldots,n$) of same product type u to the m machines by constraint programming. Be-

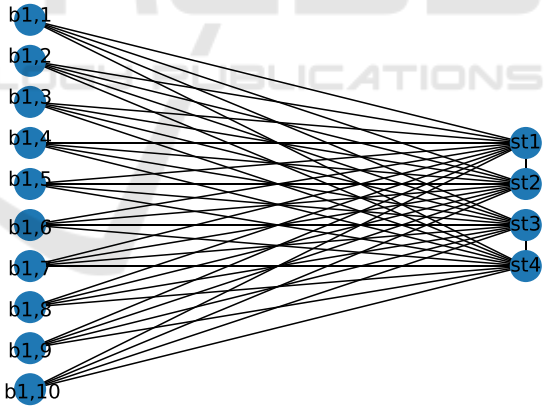

Figure 1: Bipartite graph of 10 batches $b_{u,j}$ ($j = 1,\ldots,10$) to be processed on 4 work stations st_i, $i = 1,2,3,4$. The processing times per product type $u = 1,2,3$ are 6, 8, 12 time units on the work stations $i = 1,2$ and double on the work stations $i = 3,4$. Note that the batches with types $u = 2,3$ are omitted from the figure for compactness!

Fünfzig, C., Wallrath, R. and Hubert, S.
Efficient Batch Assignment for Parallel-Machine Production Scheduling.
DOI: 10.5220/0012320800003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 268-274
ISBN: 978-989-758-680-4; ISSN: 2184-433X

sides the machine parameters, there can be additional changeover constraints on the same and between different machines so that the problem is not separable, see Figure 1 marked by a line between machines with 8 time units changeover time for a different type on the other machine and 0 time units for the same type. Especially, the problem can in general not be solved one single machine after another. In practice, further processing stages can follow after the first parallel stage like a filling stage for example.

This article analyzes different direct models for the assignment and scheduling of parallel machines with one solve, using the CP-SAT constraint solver in Google OR-Tools (Perron and Didier, 2023), where we aim for extensible models and minimum-makespan solutions. It does not consider any problem decomposition with several solves as in (Tran et al., 2016)(Hooker and Ottosson, 2003).

There is a large body of previous work on parallel machine scheduling ranging from heuristics over MILP approaches, which is beyond the scope of this paper as we aim for explaining model and solver performance of constraint models. (Da Col and Teppan, 2019) compares published benchmarks with different numbers and lengths of jobs for the job shop scheduling problem and different models without or with interval variables on two constraint solvers: IBM CP-Optimizer and CP-SAT. Conditional interval variables for scheduling have been introduced in (Laborie and Rogerie, 2008) together with their handling by the IBM CP-Optimizer.

Cardinality constraints for assignment have been considered in *exactly-n* and *atmost-n* (Sinz, 2005) (Heule, 2020) in boolean formulation. In integer constraint programming, these can be formulated as sum constraints of integers (Trick, 2003). Some constraint programming systems offer set variables with cardinality constraints (Gervet, 2006) (Benoist et al., 2011), which is translated internally into low-level constraints. Similarly, (Baxter et al., 2016) proposes special symmetry declarations inside the modeling language to override the variable indexing (Marriott et al., 2008), which are then exploited by the solver.

(Perron and Furnon, 2023) describe the general assignment problem with linear costs for CP-SAT and MILP and demonstrate grouping of assignment entities. The combination of assignment and scheduling like in our article is still unclear. To the best of our knowledge, the efficient restricted schemes in our article and a comparison for the assignment and scheduling problem have not been described elsewhere.

Section 2 states the problem and the notation used throughout the article. We present the efficient restricted assignment schemes for same-parameter batches and its formulation via constraints in Section 3, analyze their number of different feasible assignments and give results from our experimens with the SAT-based constraint solver CP-SAT (Section 4). Finally, we give conclusions in Section 5.

2 PROBLEM STATEMENT AND PREVIOUS WORK

First we give the notation used throughout the article. For each machine with index i ($i = 1,\ldots,m$) there is a boolean $S_{i,j}$ for the batch with index j ($j = 1,\ldots,n$) of same parameters (determined by volume and product type u). Note that we omit the parameters from the notation of batches in the following for shortness as the schemes apply to each of the same parameter sets similarly. If $S_{i,j} = 1$, then batch index j is assigned to the machine with index i. The indexing of all booleans $S_{i,j}$ is depicted in Figure 2, which is used for the description of all assignment schemes.

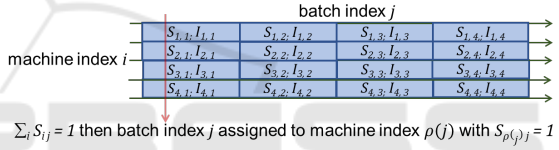

Figure 2: Variables for $n = 4$ batches (columns) to $m = 4$ machines (rows).

Additionally, $I_{i,j}$ is the time interval $[s(I_{i,j}), e(I_{i,j})[$ for processing on machine i with $s(I_{i,j})$, $e(I_{i,j})$ integers. Note that time intervals with $s(I_{i,j}) \geq e(I_{i,j})$ are empty. Each machine i can process only one job at a time (disjunctive processing), i.e., all assigned time intervals $\{I_{i,j} : j = 1,\ldots,n \wedge S_{i,j} = 1\}$ must be non-overlapping. Disjunctive processing can be achieved by a global constraint (Carlier, 1982, disjunctive)(Baptiste et al., 2012, NoOverlap) for each machine index i

$$\text{NoOverlap}(I_{i,j} : j = 1,\ldots,n \wedge S_{i,j} = 1)$$

The assignment boolean and the time interval are represented as a pair $(S_{i,j}; I_{i,j})$, which is called an optional interval in CP-SAT and other constraint solvers. Intervals $I_{i,j}$ with $S_{i,j} = 0$ are non-active (aka. non-assigned), and such intervals are simply ignored by the solver. Intervals $I_{i,j}$ with $S_{i,j} = 1$ are active.

Like in the general assignment problem, we have to assign each of the n batches to exactly one of the m machines.

$$\sum_{i=1}^{m} S_{i,j} = 1 \quad \text{for all } j\ (j = 1,\ldots,n) \qquad \text{(C.1)}$$

We denote by function ρ the machine index $\rho(j) := i$ (row) with $S_{\rho(j),j} = 1$ for the batch b_j (column). Additionally, let $z_i := \sum_j S_{i,j}$ be the number of batches assigned to machine index i. For an example with $n = 4$ batches on $m = 4$ machines see Figure 3.

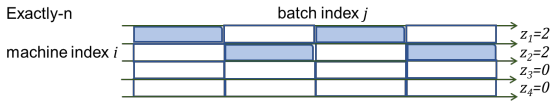

Figure 3: General assignments (**Exactly-n** and **Exactly-n Ordered**) for an example with $n = 4$ batches (columns) on $m = 4$ machines (rows) with $z_1 = z_2 = 2$, $z_3 = z_4 = 0$.

The general assignment scheme in Figure 3 allows m^n feasible assignments ρ, which is an exponential number in n. In our application, the assignment determines the active intervals on each machine with index i, but only their number z_i and not their indices matter. We show in our solving experiments in Section 4 that this general assignment scheme **Exactly-n** is not efficient.

3 ASSIGNMENT SCHEMES

Batches of same parameters are indistinguishable in scheduling, and they create solutions with just permuted indices. To avoid such index permutation (Gent et al., 2006) of time intervals, we order the non-overlapping intervals on machine i:

$$e(I_{i,j-1}) \leq s(I_{i,j}) \quad \text{for all } j \geq 2$$

In this way, we add precedence constraints on machine i based on index order.

We can also constrain the time intervals in **Exactly-n** further by ordering the end times of the overlapping time intervals on *possibly different* machines $\rho(j)$:

$$e(I_{\rho(j-1),j-1}) \leq e(I_{\rho(j),j}) \quad \text{for all } j \geq 2$$

We call this strengthened scheme **Exactly-n Ordered**.

As it appears in the results, it is more efficient and direct to restrict the assignments themselves to blocks per machine, as shown in Figure 4.

Left-Most Blocks

We additionally restrict the assignments on each machine i to have smallest indices, beginning with machine $i = 1$, where assigned batches form left-most blocks on each machine (Figure 4).

For a formulation with only linear constraints, let boolean variable $seq_{i,j}$, $seq_{i,j} = 1$ iff $S_{i,j-1} \neq S_{i,j}$, i.e., the assignment changes from column $j-1$ to column j ($j = 2, \ldots, n$). Then the sum $\sum_{j=2}^{n} seq_{i,j}$ gives the total number of boolean changes on machine i (row). The variables $S_{i,k}$ and $seq_{i,j}$ are connected by the following constraints:

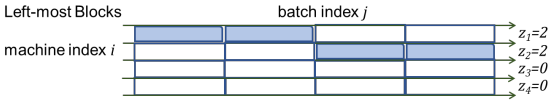

Figure 4: Restricted assignments (**Left-most Blocks** and **Left-most Blocks Table**) for an example with $n = 4$ batches (columns) on $m = 4$ machines (rows) with $z_1 = z_2 = 2$, $z_3 = z_4 = 0$. The row sums are the same as in Figure 3 but form left-most blocks on each machine.

$$\forall k \geq 2 \quad S_{1,k-1} \geq S_{1,k} \qquad \text{(C.2)}$$

row $i = 1$ starts with block (column $k = 1$)

$$\forall k \geq 2 \, \forall i \geq 2 \quad S_{i,k} = 1 \text{ if } S_{i-1,k-1} = 1 \wedge S_{i-1,k} = 0 \qquad \text{(C.3)}$$

block in row i is adjacent to block in the previous row $i-1$

$$\forall i \quad S_{i,1} + \sum_{j=2}^{n} seq_{i,j} + S_{i,n} \in \{0,2\} \qquad \text{(C.4)}$$

see cases for $\sum_{j=2}^{n} seq_{i,j}$ below

We check the three cases below to convince us that (C.4) is satisfied.

$(S_{i,k})_k = 1..10..0$ or $0..01..1$ iff $\sum_{j=2}^{n} seq_{i,j} = 1$

$(S_{i,k})_k = 0..01..10..0$ iff $\sum_{j=2}^{n} seq_{i,j} = 2$

$(S_{i,k})_k = 0..0$ or $1..1$ iff $\sum_{j=2}^{n} seq_{i,j} = 0$

In all other cases $\sum_{j=2}^{n} seq_{i,j}$ is larger than 2 and thus excluded by the constraints.

Left-Most Blocks Table

For a formulation with the global constraint *AllowedAssignments*, we list all possible blocks of ones as allowed variable assignments for $(S_{i,j})_j$, resulting in scheme (**Blocks Table**).

Constraints (C.2) and (C.3) restrict assignments to smallest indices as above in **Left-most Blocks**, resulting in the scheme **Left-most Blocks Table**. Note that **Left-most Blocks Table** is a different implementation of **Left-most Blocks** without the additional boolean variables $seq_{i,j}$ for forming blocks. The allowed assignments are listed in the table, and the dependencies between rows are achieved by constraints (C.2) and (C.3).

4 RESULTS

In this section, we compare the schemes by the number of feasible assignments and based on solver experiments with OR-Tools CP-SAT.

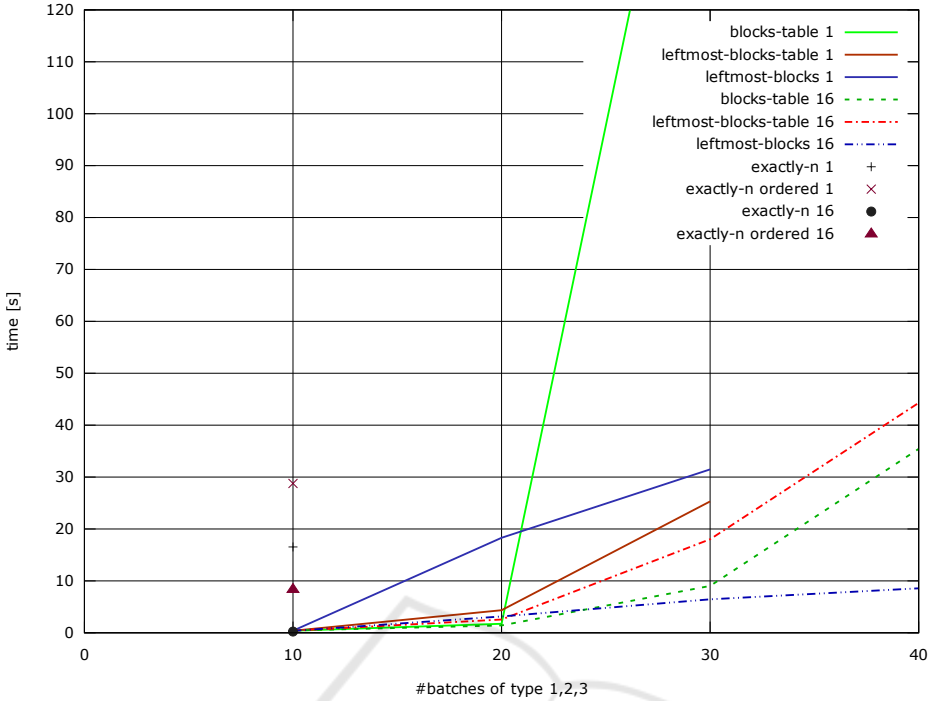

Figure 5: Solving times for finding an optimal solution without proving optimality. (AMD Ryzen 9–5950, Windows 10 Professional, OR-Tools CP-SAT version 9.7.2996 with given number of workers 1 or 16).

Feasible Assignments. The schemes **Exactly-n**, **Exactly-n Ordered** have the most assignments with same numbers z_i. **Exactly-n Ordered** additionally reduces the number of solutions by ordering interval end times over all $m = 4$ machines.

Blocks Table: still keeps several assignments with same numbers z_i as it assigns blocks to the machines $2, 3, \ldots$ without prescribing their orders. If $k \leq n$ is the number of actually used machines then there are $k! \cdot$ **#Left-most Blocks**.

In this example with one state, **Left-most Blocks** has only 8 feasible assignments, and their exact number is given by the number of ordered sums $K(n,m) := \sum_{i=1}^{m} z_i$ of m numbers $0 \leq z_i \leq n$. Bounds for this number $K(n,m)$ can be calculated from the ordered partition number $G(n,k)$ of $k \leq m$ numbers $1 \leq z_i \leq n$, which is $G(n,k) = \binom{n-1}{k-1}$ (Matousek and Nesetril, 2009). Placing the non-zero numbers on the m machines gives $K(n,m) = \sum_{k=1}^{m} \binom{m}{k} G(n,k)$. Then for a lower bound

$$\binom{m}{\lfloor m/2 \rfloor}\binom{n-1}{\lfloor m/2 \rfloor - 1} \leq \sum_{k=1}^{m} \binom{m}{k}\binom{n-1}{k-1},$$

which results from any large summand in the sum. For an upper bound, we assume $n > 2m$ as in applications

$$\begin{aligned} \sum_{k=1}^{m} \binom{m}{k}\binom{n-1}{k-1} &\leq m \binom{m}{\lfloor m/2 \rfloor} \max_{k=1}^{m} \binom{n-1}{k-1} \\ &\leq m^{\lfloor m/2 \rfloor + 1} (n-1)^{m-1} \end{aligned}$$

which is polynomial in n with a fixed number m of machines.

Solver Experiments. We compare the schemes for computing a minimum-makespan production schedule for 3 product types with different processing times on the machines. We consider solutions for the example problem in Figure 1, where there are additional waiting times between processing different types on machines 1, 2 and machines 3, 4 as they arise from a shared material feeder.

The experiments were performed on AMD Ryzen 9–5950 (16 cores, 96GB RAM), with OR-Tools version 9.7.2996 using the Python API on Windows 10 Professional. Solver settings are default with presolving and 1 (sequential) or 16 (parallel portfolio) workers. Figure 5 shows the solving times without proving it optimal for a solution with minimum makespan 88, 174, 262 and 348 respectively.

Exactly-n and **Exactly-n Ordered**, only performed for 10 batches per type, are orders of magnitude slower than the block schemes. With both, there are up to $\binom{n}{z_i}$ feasible assignments for given z_i, so that the active intervals $(I_{i,j} : \sum_j S_{i,j} = z_i)$ can have arbitrarily dispersed indices on each machine i. It seems not easily possible for the solver to extend partial solutions with these schemes. Additional ordering of intervals by end time does not help and makes it even slower.

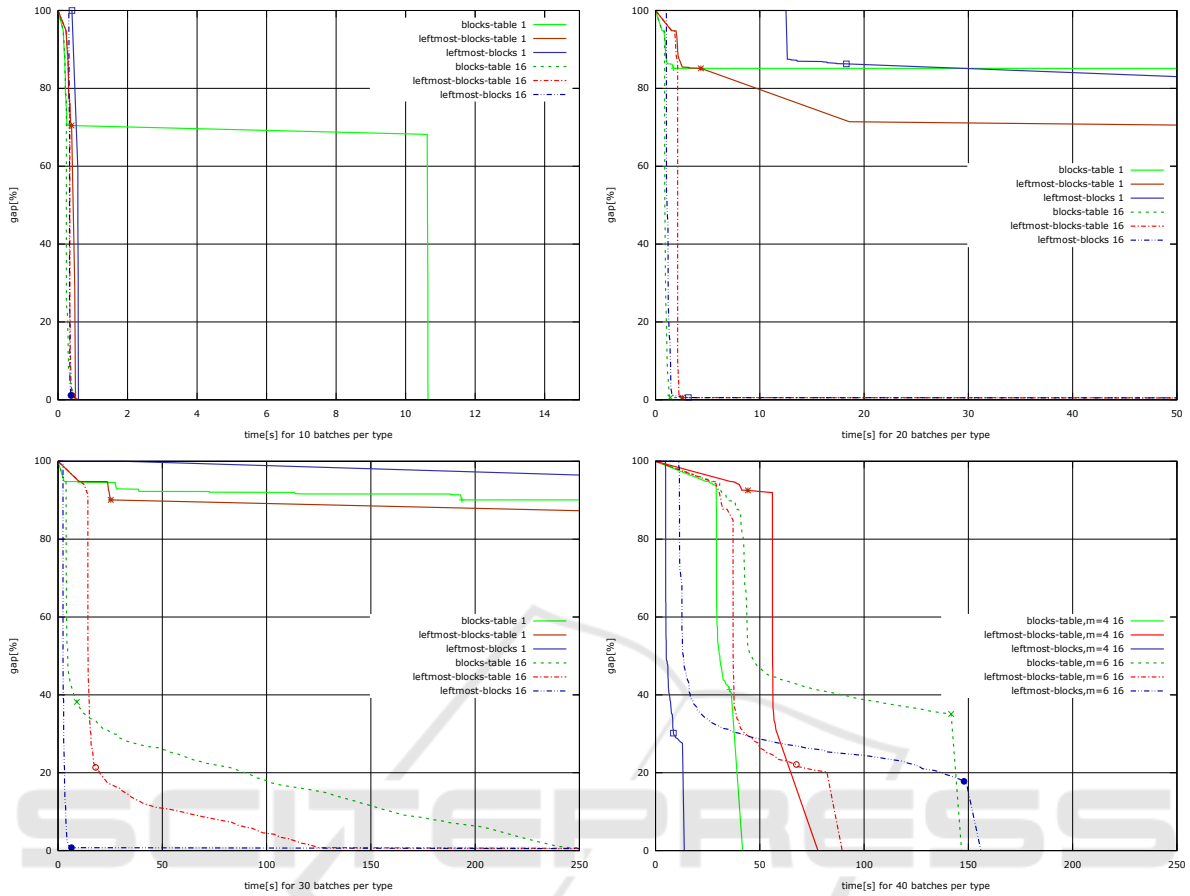

Figure 6: Optimality gaps for $n = 10, 20$ (top), $n = 30, 40$ (bottom) till timeout 1000 seconds or proving optimality. Points on the curves indicate when the optimal solution is found. (AMD Ryzen 9–5950, Windows 10 Professional, OR-Tools CP-SAT version 9.7.2996 with given number of workers 1 or 16).

With 1 worker the schemes **Left-most Blocks** and **Left-most Blocks Table** are the fastest, whereas **Blocks Table** gets slow for 30 or more batches. For smaller numbers of batches, the search with objective optimization seems to select within the larger number of feasible solutions. With 16 workers all schemes are much closer together with **Left-most Blocks** being the fastest, especially for large numbers of batches. Again, **Blocks Table** is fast for all numbers of batches.

We also looked into how lower bounds and objective values evolve during the optimization process, Figures 6 and 6 show the optimality gaps beginning with the example $n = 10$ up to $n = 40$.

As can be seen, with 1 worker the lower bounds are only slowly improving, **Blocks Table** being worst. With 16 workers and larger number of batches, all schemes are much closer together, with **Left-most Blocks** being the best. The additional booleans in the formulation seem to be beneficial during the portfolio search. If the number of machines ($m = 6$ instead of $m = 4$ in our example) gets larger, then **Left-most Blocks Table** is better than **Blocks Table**, as permutations of machines with **Blocks Table** allow more feasible assignments.

5 CONCLUSION

In this article, we compared three different batch assignment schemes for parallel-machine scheduling. As batches with identical parameters occur in a large number, this is a very important part of the production constraint model. As an application example, we show a filling stage connected to the processing stage in the constraint model or added as a filling heuristic (Jaehn and Pesch, 2019)(Pinedo, 2016, dispatching rules). Figure 7 shows the Gantt charts for 1 filling station in our example with 3 product types of $n = 10$ batches each and $m = 4$ machines.

In our experiments with the solver CP-SAT, the general schemes **Exactly-n** and the further con-

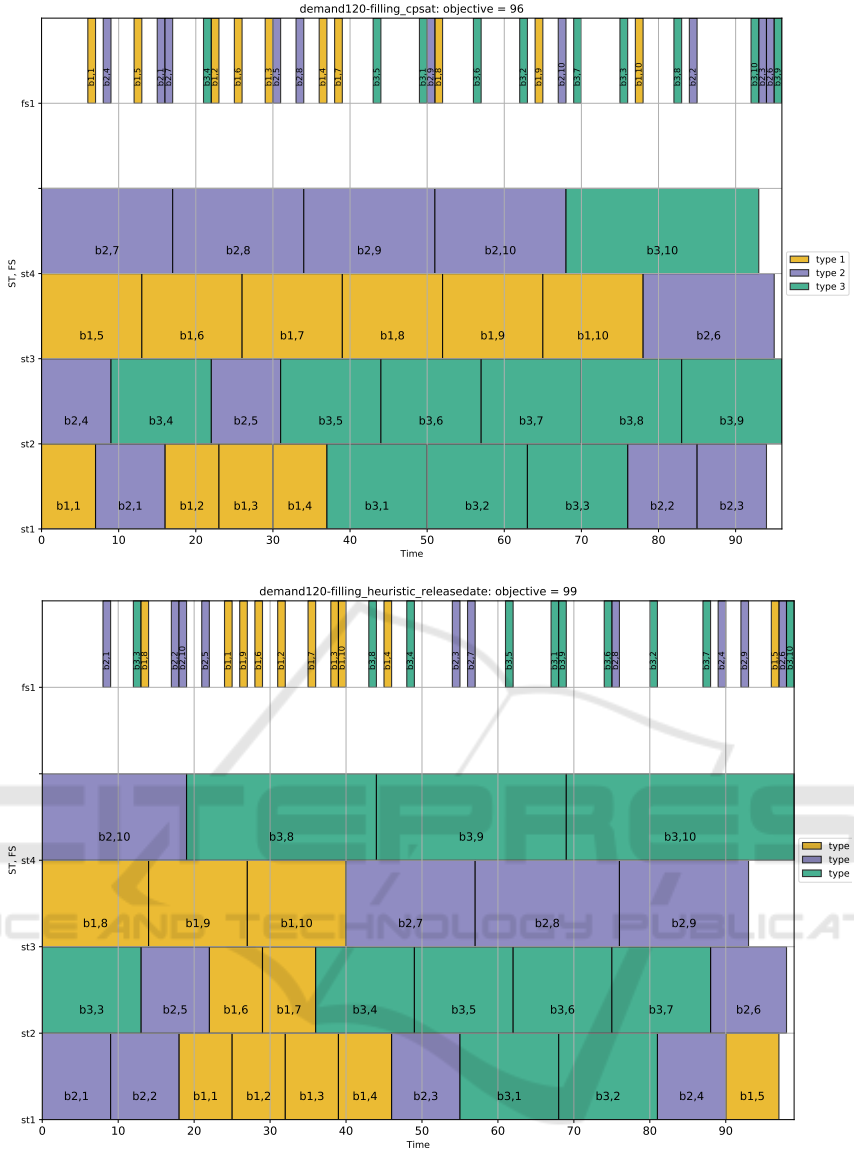

Figure 7: Gantt charts for the problem in Figure 1. Solved to minimum makespan with filling by CP-SAT (top) and by a filling heuristic *earliest-release-date-first* (bottom) after the processing stage by CP-SAT. Note that the production schedules are different, and the overall minimum makespan is 3 time units smaller due to waiting times for filling. Furthermore, there are normally several schedules of minimum makespan.

strained **Exactly-n Ordered** are very slow in terms of solving time. For practical, non-separable problems, we presented other schemes with significantly smaller, polynomial number of feasible assignments and much better solving times for a minimum-makespan schedule.

The scheme **Left-most Blocks** performs best especially for larger number of batches and for lower bounds determination during solving. **Blocks Table** performs well for optimization with a small number of machines. With **Blocks Table** all pre-generated allowed blocks are provided in the *AllowedAssignments* global constraints as arguments. If the number of machines gets larger, then **Left-most Blocks Table** performs better.

In summary, keep the activation pattern in disjunctive constraints similar during solving like in all restricted assignment schemes and prefer boolean constraint formulations for determining the activation pattern (**Left-most Blocks** and **Left-most Blocks Table**).

The **Exactly-n Ordered** scheme has non-decreasing interval end times over all machines by constraint. Such an explicit sorting is not neces-

sary for correctness of parallel-machine scheduling but it is an interesting property for decomposition approaches. As future work, we are interested in decomposition approaches for different path possibilities (direct or via additional storage tanks to several filling stations).

REFERENCES

Baptiste, P., Pape, C., and Nuijten, W. (2012). *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. International Series in Operations Research & Management Science. Springer International Publishing.

Baxter, N., Chu, G., and Stuckey, P. J. (2016). Symmetry declarations for minizinc. In Parry, D., editor, *Proceedings of the 39th Australasian Computer Science Conference – ACSC 2016*.

Benoist, T., Estellon, B., Gardi, F., Megel, R., and Nouioua, K. (2011). Localsolver 1.x: a black-box local-search solver for 0-1 programming. *4OR*, 9:299–316.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47.

Da Col, G. and Teppan, E. C. (2019). Industrial size job shop scheduling tackled by present day cp solvers. In *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30 – October 4, 2019, Proceedings*, page 144–160, Berlin, Heidelberg. Springer-Verlag.

Gent, I. P., Petrie, K. E., and Puget, J.-F. (2006). Chapter 10 - symmetry in constraint programming. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier.

Gervet, C. (2006). Chapter 17 - constraints over structured domains. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 605–638. Elsevier.

Heule, M. (2020). Representations for automated reasoning. Course *Advanced Topics in Logic: Automated Reasoning and Satisfiability*. Carnegie Mellon University.

Hooker, J. and Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, 96:1436–4646.

Jaehn, F. and Pesch, E. (2019). *Ablaufplanung: Einführung in Scheduling*. Springer International Publishing, 2nd edition.

Laborie, P. and Rogerie, J. (2008). Reasoning with conditional time-intervals. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference (FLAIRS 2008)*, Berlin, Heidelberg. AAAI Press, Palo Alto, California USA.

Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P. J., de la Banda, M. G., and Wallace, M. (2008). The design of the zinc modelling language. *Constraints*, 13(3):229–267.

Matousek, J. and Nesetril, J. (2009). *Invitation to Discrete Mathematics*. Oxford University Press, 2nd edition.

Perron, L. and Didier, F. (2023). CP-SAT solver using SAT methods. http://developers.google.com/optimization/cp.

Perron, L. and Furnon, V. (2023). OR-Tools Guides: Assignment. http://developers.google.com/optimization/assignment.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 5th edition.

Sinz, C. (2005). Towards an optimal cnf encoding of boolean cardinality constraints. In van Beek, P., editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 827–831. Springer International Publishing.

Tran, T. T., Araujo, A., and Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95.

Trick, M. A. (2003). A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118.