Preemptive jobshop scheduling with maximum workload constraints

Tanguy Terrien[1]
LAAS-CNRS, Université Paul Sabatier, Toulouse, France
IRIT, Université Paul Sabatier, Toulouse, France

Cyrille Briand
LAAS-CNRS, Université Paul Sabatier, Toulouse, France

Abstract

Optimizing schedules in real-world settings often requires considering maximum workload constraints, especially for human resources, to ensure regulatory compliance, impose rest periods, and/or level the workload over the working horizon. This paper focuses on this particular problem within the domain of preemptive jobshop scheduling, a central problem in scheduling theory. Preemption is particularly relevant when human resources are involved, allowing personnel to flexibly switch between tasks. It also offers theoretical insights as a relaxation of non-preemptive problems. The main contribution of this paper is a Constraint Programming approach designed to handle maximum workload constraints in a preemptive setting, without decomposing activities into unit-duration tasks (which may be computationally prohibitive). The experimental results demonstrate the effectiveness of our approach on a set of examples, highlighting its performance compared to a well-known industrial solver, IBM's CP Optimizer.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

Keywords and phrases Constraint Programming, Scheduling, Maximum workload constraints

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements This work was supported by the French National Research Agency (ANR) under the project **HIS**^{3} ANR-22-CE10-0012-05.

1 Introduction

Optimizing schedules in real-world settings requires considering maximum workload constraints, especially for human resources, to ensure regulatory compliance, prevent fatigue, and maintain productivity [9]. Neglecting these constraints can lead to significant legal penalties, reduced productivity, increased errors, or significant employee turnover. Conversely, ensuring adequate rest and manageable workloads improves employee well-being and enhances overall operational safety and efficiency. Sectors like project management, healthcare (e.g., nurse scheduling) and transportation (e.g., train driver scheduling) face this kind of regulations, which make the integration of such workload and rest time rules a complex aspect of scheduling. General overviews of scheduling theory and manufacturing processes also underscore the importance of these considerations [9].

Scheduling approaches based on constraint programming (CP) now enable the efficient resolution of industrial problems, from medium to large size, involving complex and heterogeneous constraints (e.g., see [4]). CP frameworks allow integrating diverse constraints, including constraints arising from the involvement of human operators in production processes. However, modeling the preemptive nature of human activities remains challenging for

[1] Corresponding author

CP-based approaches. Indeed, in many real-world scenarios, human operators may interrupt started tasks. This preemptive behavior introduces significant combinatorial complexity in determining schedules. The resulting increase in the solution space often leads to a degradation in the performance of constraint-based methods when applied to such highly dynamic and flexible scheduling environments.

To account for human-centered concerns in scheduling while avoiding the above mentioned combinatorial issue, we introduce a new class of constraints that model rest-time requirements for operators over specific time intervals: the **MaxW** constraints (short for **Max**imum **W**orkload constraints), presented in Section 3. The contributions of this paper are outlined below: (i) we introduce a new class of workload constraints for operators, which generalize practical requirements often found in industry, such as the prohibition of working on two consecutive shifts; (ii) we propose a constraint programming model for the preemptive jobshop problem with these constraints, building on the Mistral solver and adapting its propagation algorithms to handle these constraints efficiently; (iii) we also present a more extensive model using IBM® ILOG CPLEX Optim Studio (CPO) [4], used as a baseline for comparison with our Mistral modeling; (iv) finally, building on existing pJSP instances (preemptive Jobshop scheduling Problem), we generate a set of benchmark instances incorporating workload constraints, varying the density of mandatory rest periods.

The remainder of the paper is organized as follows: Section 2 reviews related work, Section 3 formalizes the problem and the proposed constraints, Section 4 details the modeling approaches with two different solvers (Mistral and CPO), Section 5 presents and analyzes the experimental results, and Section 6 concludes and outlines directions for future work.

2 Brief state-of-the-art

The resource-constrained project scheduling problem (RCPSP) provides a foundational framework to model limited resource availability in scheduling. When human resources are involved, traditional RCPSP formulations are extended to incorporate specific workload constraints such as maximum working hours or minimum rest periods between tasks [4].

Mathematical Programming, particularly Mixed-Integer Linear Programming (MILP), is also common for modeling workload constraints, as seen in nurse rostering problems, where integer programming techniques are applied [10] or column generation-based heuristic approaches [11]. While powerful, exact solutions for large instances can be computationally costly. Metaheuristics are widely employed for intractable large-scale problems such as complex scheduling environments, like nurse rostering [1, 5].

In this paper, the focus is on the Jobshop Scheduling Problem (JSP). It is a classic NP-hard problem that is focused on processing jobs on machines. Introducing human operators with specific constraints makes the problem significantly more complex. Each operation may require not only a machine but also an operator with limited working hours and mandatory rest periods. While the general JSP has been extensively studied, research papers explicitly integrating human resources are scarce. In [7] a state-of-the-art and a branch-and-bound algorithm are proposed to deal with operators' unavailability constraints. A so-called filter-and-fan based heuristic is also proposed for scheduling in flexible job shops under workforce constraints in [8].

The preemptive Job Shop Scheduling Problem (pJSP) further complicates JSP by allowing operation interruptions. While this can lead to better makespan, it significantly complicates scheduling logic. The pJSP maximum workload constraints is a generalization of the already NP-hard pJSP (considering makespan minimization). It is therefore also NP-hard. Adding

maximum workload constraints on human resources introduces a new dimension, requiring careful consideration of breaks, shift limits, and rest periods for assigned operators. This intertwines personnel scheduling's resource constraints with job shop operations' intricate dependencies. Research into online printing shop scheduling, which also involves flexible job shop problems with resumable operations and machine unavailability, has employed CP models [6]. While their work focuses on makespan minimization without explicit maximum workload constraints, these studies provide a strong foundation for modeling and solving pJSP with additional complexity, which is directly applicable to our context. Relevant work by Juvin et al. [3] presents an efficient CP approach designed for the pJSP. This work does not take into account maximum workload constraints, but offers the advantage of avoiding the combinatorial enumeration of all shifts. We show below how this work can be extended to MaxW constraints.

3 Considered problem

A **MaxW** constraint is defined as a triplet $\left(\delta_{uv}^k, k, [u, v]\right)$, where k is an operator (or machine), $[u, v]$ is a time interval, and δ_{uv}^k is the minimum required number of rest shifts (i.e., time units not allocated to any task) for operator k within this interval. We consider the Maximum Workload preemptive Jobshop Scheduling Problem (MaxW-pJSP). Work is defined by a set of jobs $\mathcal{J}$, where each job $J_i \in \mathcal{J}$ is composed of a set of n_i tasks $t_{i,j}$ for $j \in \{1, ..., n_i\}$ (i indexes the job). We consider K operators, indexed from 0 to $K-1$. Each task $t_{i,j}$ has a processing time $p_{i,j}$, an earliest starting date $s_{i,j}$ and an latest completion date $e_{i,j}$. Furthermore, each task is assigned to an operator that must execute it. Note that we consider the preemptive version of the problem, i.e. an operator can interrupt a task anytime and resume it later. To model workload regulation, we generate for each operator k a set of maximum workload constraints under the form $\left(\delta_{uv}^k, k, [u, v]\right)$ (see Section 5.1 for details on how they were generated to reflect a realistic production environment). The objective is to minimize the makespan, denoted by $C_{\max}$, which represents the total completion time of the schedule. Our scheduling problem includes several types of constraints, which together ensure the feasibility and coherence of the schedule. Each task $t_{i,j}$ has a duration constraint, enforcing that any task is fully executed. A task may also have a release date and a due date to respect. Every task is assigned a specific unique operator. All tasks within a job are ordered (it is called precedence constraints, i.e. $t_{i,2}$ can only start when $t_{i,1}$ is finished, etc). All working shifts and rest shifts of a same operator should not overlap. These constraints define the pJSP, to which we add MaxW constraints.

4 Modeling

To improve computational efficiency, we avoid explicitly modeling every working shift by leveraging the Mistral solver and its specialized PreemptiveNoOverlap constraint [3]. This constraint bounds task execution to ensure the operator has sufficient time for all preemptive tasks within the horizon. Mistral's tailored propagation, including overload checks, efficiently solves the **pJSP** with makespan minimization. An explicit shift-based schedule is then derived using Jackson's polynomial-time algorithm [2]. This solution is guaranteed to be optimal. This approach significantly reduces the number of variables and constraints. We propose a new Mistral-based model for this problem.

4.1 Mistral model

In the following, if not specified, consider $i \in \{0, ..., |\mathcal{J}|-1\}$, $j \in \{0, ..., n_i\}$, $k \in$ Operators. q indexes the SubIntervals of a MaxW constraint (explained in Figure 1), i.e. and $q \in$ SubIntervals(MaxW). We write $\mathcal{C}_k$ the set of MaxW constraints of operator k, tasks_k the set of tasks of operator k (which is fixed beforehand).

$$\min C_{\max}$$

$$s_{i,j} \in [0, \text{ub} - p_{i,j}],\ e_{i,j} \in [p_{i,j}, \text{ub}] \qquad \forall i, j \qquad \text{(V1)}$$

$$C_{\max} \in [0, \text{ub}] \qquad \text{(V2)}$$

$$\text{d}_{kq} \in [0, |q|] \qquad \forall k, \forall q \qquad \text{(V3)}$$

$$e_{i,j} \geq s_{i,j} + p_{i,j} \qquad \forall i, \forall j \qquad \text{(C1)}$$

$$s_{i,j+1} \geq e_{i,j} \qquad \forall i, \forall j \qquad \text{(C2)}$$

$$s_{i,0} = 0 \quad \text{and} \quad e_{i,n_i} = C_{\max} \qquad \forall i \qquad \text{(C3)}$$

$$\textsc{Pre.NoOverlap}\Big(\{(s_{i,j}, e_{i,j}, p_{i,j})\} \bigcup \{(start_q, end_q, d_{kq})\}\Big) \qquad \forall i, \forall j \in \text{tasks}_k, \forall k, \forall q \qquad \text{(C4)}$$

$$\sum_{q \in [u_z, v_z]} d_{kq} \geq \delta_z \qquad \forall k, \forall q, \forall (\delta_z, u_z, v_z) \in \mathcal{C}_k \qquad \text{(C5)}$$

$$d_{kq} \leq max(\delta) \qquad \forall k, \forall (\delta_z, u_z, v_z) \in \mathcal{C}_k, \forall q \in [u, v] \qquad \text{(C6)}$$

Variables: For each task j of each job i, we define a start time variable and an end time variable (V1), where ub is an upper bound on the schedule length. The objective is to minimize the makespan, denoted by $C_{\max}$ (V2), which represents the total completion time of the schedule.

Additional variables: To handle workload constraints, we divide the time horizon into subintervals. Every point in time at which a MaxW constraint may begin or finish defines a boundary of a subinterval. As a result, the full scheduling horizon is partitioned into a sequence of non-overlapping time windows defining potential starts and ends of a subinterval. Note that these subintervals cover every shift. We then introduce, for each operator k and each subinterval q, a rest duration variable d_{kq} representing the amount of rest shifts allocated to operator k within subinterval q. For all k and q, d_{kq} is non-negative and is bounded from above by the size of interval q (V4) and by the biggest value of δ of all MaxW constraints that contain interval q (C6).

Let us give an example. Consider a unique operator with two MaxW constraints (3, 0, 8) and (2, 4, 10). The MaxW constraints, computed subintervals and their associated duration variables are depicted in the Figure 1:

Constraints: Each task $t_{i,j}$ has a duration constraint, enforcing that any task is fully executed (C1). Every task is assigned a specific unique operator (in (C4), tasks_k define all tasks assigned to operator k). Tasks within a same job are linked by precedence constraints (C2). To ensure that no two tasks of a same operator overlap, a preemptive non-overlap constraint is imposed for each operator k (C4), to its set of actual tasks and to the additional rest variables. Lastly, the model ensures that maximum workload constraints are verified. Indeed, for each operator k and each MaxW constraint defined by a triple $(\delta^k_{u,v}, k, [u, v])$, the model identifies the set of rest subintervals that are contained in interval $[u, v]$. The total duration of rest assigned within these subintervals must be at least δ (C5), thereby ensuring compliance with operator workload limitations.

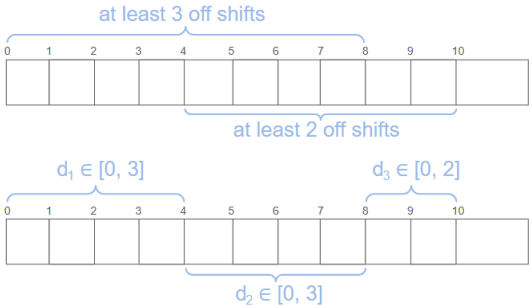

Figure 1 Two MaxW constraints, their subintervals and associated duration variables

By enforcing a PreemptiveNoOverlap constraint on all working tasks and all rest shifts for each operator, we ensure that tasks can be scheduled preemptively without conflict. Note that the original Mistral solver does not support variable task durations in the PreemptiveNoOverlap constraint. We adapted the propagation algorithm to accommodate our requirements. This improvement is soon to be added to the open source code of Mistral, that can be found on GitHub). Thanks to the adapted propagation algorithm, variables $s_{i,j}$, $e_{i,j}$ and d_{kq} are computed by the solver in a way that guarantees that a feasible preemptive schedule exists and that it is optimal in terms of makespan.

4.2 CPO model

As a comparison to our approach, we also implemented a model for the MaxW-pJSP using CPO, a leading CP solver for scheduling [4]. When preemption is forbidden, the modeling process is straightforward: each task is represented as a continuous block of fixed size, and the solver's main task is to determine the order of these blocks. In such cases, advanced algorithms like edge-finding or detectable precedences, as described in [12], are particularly effective. However, when preemption is allowed, tasks can alternate for a single operator, notably increasing the combinatorial complexity. This requires introducing a decision variable for every shift of every task, making it more challenging for the solver to propagate constraints effectively. The model is the following (for notations, see beginning of subsection 4.1). Also we write $c_{kz} = (\delta_{kz}, u_{kz}, v_{kz}) \in \mathcal{C}_k, \quad \forall z \in \{1, ..., |\mathcal{C}_k|\}$.

$$\min \quad \max_{i,j} \left(\text{endOf}(W_{i,j,p_{ij}}) \right)$$

$$\text{interval } W_{ij\ell}, \text{ size} = 1 \qquad \forall i, \forall j, \forall \ell \in \{1, ..., p_{ij}\} \qquad \text{(V1)}$$

$$\text{interval } O_{kc_{kz}\lambda}, \text{ optional }, \text{size} = 1 \qquad \forall k, \forall c_{kz}, \forall \lambda \in \{1, ..., \delta_z\} \qquad \text{(V2)}$$

$$O_{kc_{kz}\lambda} \subseteq [u_{kz}, v_{kz}] \qquad \forall k, \forall c_{kz}, \forall \lambda \in \{1, ..., \delta_z\} \qquad \text{(C1)}$$

$$\text{EndBeforeStart}(W_{ij,\ell-1}, W_{ij,\ell}) \qquad \forall i, \forall j, \forall \ell \in \{2, ..., p_{ij}\} \qquad \text{(C2)}$$

$$\text{EndBeforeStart}\left(W_{i,j,p_{ij}}, W_{i,j+1,0}\right) \qquad \forall i, \forall j \in \{0, ..., n_i - 1\} \qquad \text{(C3)}$$

$$\text{NoOverlap}\left([W_{ij\ell}]_{\forall i,j,\ell} \cup [O_{kc\lambda}]_{\forall c,\lambda}\right) \qquad \forall k, j \in \text{tasks}_k \qquad \text{(C4)}$$

$$\sum_{\substack{y=1...|\mathcal{C}_k| \\ \lambda=1...|\delta_y|}} \text{Presence}(O_{kc_{ky}\lambda}).(O_{kc_{ky}\lambda} \in [u_{kz}, v_{kz}]) \geq \delta_z \qquad \forall k, \forall c_{kz} \qquad \text{(C5)}$$

In this model, we have two *families* of decision variables. For each task j of each activity i, we add as many unit-length interval variables (V1) as the processing time of the task (indexed by ℓ). To deal with MaxW constraints, the model also features a set of interval variables O. For every MaxW constraint (δ, u, v) of operator k, we add δ unit-length *optional* interval variables (V2). If these optional variables are included in the final solution, they must be scheduled within the interval $[u, v]$ (C1). These variables allow flexibility in how rest periods are scheduled to satisfy workload regulations. The objective is still to minimize the makespan (i.e. minimize the latest end of every last shift of every task). Constraint C2 enforces sequential execution of a task: for each unit ℓ of any task ij except the last, the shift represented by variable $W_{ij\ell}$ must end before $W_{ij(\ell+1)}$ begins. Precedence constraints between successive operations of the same activity are also imposed (C3). Constraint C4 enforces a no-overlap condition, ensuring that for each operator, all its working (W) and rest (O) shifts do not overlap in time. Lastly, this model handles the max workload requirement (C5). For each operator k and each MaxW constraint c_{kz} defined over a time window $[u_{kz}, v_{kz}]$, the model requires for at least δ_z optional intervals to be present in this window.

5 Experiments

5.1 Generating instances

To run experiments on the MaxW-pJSP problem, we took pJSP instances of the literature [3], and generated MaxW constraints. The first main parameter of our algorithm to generate MaxW instances is the global desired density ($\mathcal{D}$). It expresses the overall proportion of rest days that each operator should have over the entire horizon ($\mathcal{H}$) of the instance. These rest days are distributed across multiple time intervals, which are randomly chosen but must respect some constraints. Each interval should be at least 3 shifts, and at most $0.2 * \mathcal{H}$ shifts (neither too short nor too long). Within each interval, the number of days of rest is related to the second main parameter, the local desired density ($\mathring{d}$). It expresses the overall proportion of rest days that the operator should have over any specific MaxW interval. This proportion is chosen so that its distribution is gaussian: $\mathcal{N}(\mathring{d}, \mathring{d}/4)$. This method introduces some unpredictability into when rest periods occur and how long they are. MaxW constraints are generated for each operator until the total number of rest days for all operators meet the desired global density target.

We generated MaxW constraints for all triplets (instance, $\mathcal{D}$, $\mathring{d}$), with respectively instance in {*abz5-9*} ∪ {*ft06, ft10, ft20*} ∪ {*la01-40*} ∪ {*orb01-10*} ∪ {*swv01-20*}, $\mathcal{D}$ in {0.1, 0.2, 0.3} and $\mathring{d}$ in {0.05, 0.1, 0.15, 0.2, 0.25, 0.3}, which makes a total of 78*18 = 1,404 instances.

5.2 Results

Experiments were ran on an Intel E5-2695 v4 2.1Ghz CPU with 256 GB of RAM and 36 hearts. The time limit for every instance was an hour.

Solver	Mistral	CPO	Equality
Best Solution (makespan value)	1057 (75.3%)	278 (19.8%)	69 (4.9%)

Table 1 Comparison of number of best solutions found by solvers

Table 1 sums up the number of instances where respectively the Mistral approach/the CPO approach/none found the best solution in under an hour. We observe that Mistral performed better than CPO in more than three quarters of all instances (1057 over 1404).

Only 20% of all 1404 instances were solved better by the CPO solver, and both solvers found the same lowest makespan in 69 instances.

Figure 3 in the appendix presents the number of instances were an optimal solution was found under an hour across 18 different global and local density settings for our set of 78 pJSP instances. One can see that except for variants of ft06, in all 77 problems, the Mistral solver shows optimality on at least as much instances as the CPO solver. In total, 41.5% of all 1404 instances are solved to optimality by Mistral, against only 5.3% for CPO. The approach using Mistral demonstrates a clear superiority overall. We can also observe from this table that some families of instances are harder than others. The *swv* instances are probably hard because of their sizes (very large instances), while *abz* instances have high durations of tasks. *orb* instances seem intrinsically hard.

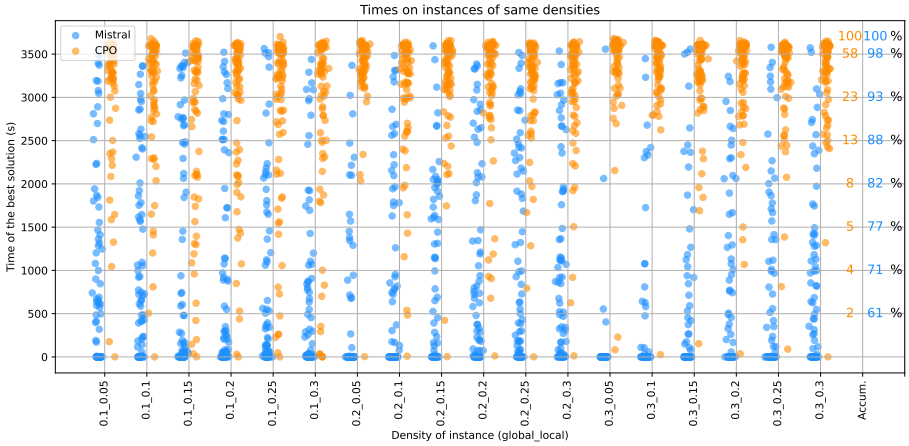

Figure 2 Time comparison for instances with same global and local densities

Figure 2 compares runtimes of both approaches, on instances grouped by identical density parameters, indicated as pairs. Each dot represents the runtime of the best solution found for a particular instance. To the right of the plot, the vertical accumulation shows, for each solver, the proportion of instances were the best solution was found before the corresponding time. From this plot, it is evident that Mistral consistently outperforms CPO across all instance densities, finding solutions significantly faster. This performance gap is especially pronounced on instances with density pairs [0.3, 0.05] and [0.3, 0.1] (i.e. many MaxW constraints with a low number of rest shits), which appear to be particularly easy for Mistral.

Overall, Mistral proves to be significantly more effective than CPO, offering better solution quality, more optimality proofs, faster computation times and scaling well to large instances.

6 Conclusion

In this work, we introduced a new class of workload constraints for operators in the context of the preemptive job shop scheduling problem. These constraints generalize commonly used rules in industrial settings, such as avoiding consecutive shifts, and aim to provide more flexible and expressive models of operator fatigue and workload balance. We proposed a constraint programming model implemented in the Mistral solver, leveraging its efficient handling of preemption, and we adapted key propagation mechanisms to support the new constraints. We also presented a CPO model used as a reference for empirical evaluation.

A set of benchmark instances was generated to test the models under varying workload constraint densities. The model implemented using Mistral proved to be significantly more efficient than the one using CPO, both in terms of computation time and effectiveness (optimality proofs).

Future work will focus on extending the proposed framework with minimum workload constraints (MinW), which complement the MaxW constraints by enforcing minimum work durations in specific time intervals. Another promising direction involves the development of hybrid decision strategies that combine disaggregated reasoning (e.g., disjunctive resource scheduling, as used in this paper) with aggregated reasoning (e.g., cumulative resource modeling) to better handle long-term planning, where exact activity timings are not yet fixed. Such approaches could enable more scalable scheduling in real-world applications.

References

1 S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*.

2 W.A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 1974.

3 Carla Juvin, Emmanuel Hebrard, Laurent Houssin, and Pierre Lopez. An efficient constraint programming approach to preemptive job shop scheduling. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*.

4 P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG. *Constraints An Int. J.*, 2018.

5 Z. Liu, Z. Liu, Z. Zhu, Y. Shen, and J. Dong. Simulated annealing for a multi-level nurse rostering problem in hemodialysis service. *Applied Soft Computing*.

6 W.T. Lunardi, E.G. Birgin, P. Laborie, D.P. Ronconi, and H. Voos. Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers Operations Research*.

7 Ph. Mauguière, J.-C. Billaut, and J.-L. Bouquard. New Single Machine and Job-Shop Scheduling Problems with Availability Constraints. *Journal of Scheduling*, 2005.

8 David Müller and Dominik Kress and. Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. *International Journal of Production Research*, 2022.

9 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.

10 H.G. Santos, T.A.M. Toffolo, R.A.M. Gomes, and S. Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 2016.

11 P. Strandmark, Y. Qu, and T. Curtois. First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem. *Computers & Op. Research*, 2020.

12 Petr Vilím, Roman Barták, and Ondrej Cepek. Extension of $O(n\log n)$ filtering algorithms for the unary resource constraint to optional activities. *Constraints*, pages 403–425, 2005.

A appendix1

	Mistral	CPO
abz5	0 (0.0%)	0 (0.0%)
abz6	2 (11.0%)	0 (0.0%)
abz7	1 (6.0%)	0 (0.0%)
abz8	0 (0.0%)	0 (0.0%)
abz9	0 (0.0%)	0 (0.0%)
ft06	15 (83.0%)	16 (89.0%)
ft10	0 (0.0%)	0 (0.0%)
ft20	15 (83.0%)	0 (0.0%)
la01	18 (100.0%)	15 (83.0%)
la02	18 (100.0%)	0 (0.0%)
la03	17 (94.0%)	0 (0.0%)
la04	14 (78.0%)	0 (0.0%)
la05	18 (100.0%)	15 (83.0%)
la06	18 (100.0%)	6 (33.0%)
la07	17 (94.0%)	0 (0.0%)
la08	18 (100.0%)	3 (17.0%)
la09	18 (100.0%)	4 (22.0%)
la10	18 (100.0%)	5 (28.0%)
la11	17 (94.0%)	2 (11.0%)
la12	18 (100.0%)	2 (11.0%)
la13	18 (100.0%)	1 (6.0%)
la14	18 (100.0%)	4 (22.0%)
la15	17 (94.0%)	1 (6.0%)
la16	0 (0.0%)	0 (0.0%)
la17	12 (67.0%)	0 (0.0%)
la18	0 (0.0%)	0 (0.0%)
la19	0 (0.0%)	0 (0.0%)
la20	1 (6.0%)	0 (0.0%)
la21	1 (6.0%)	0 (0.0%)
la22	8 (44.0%)	0 (0.0%)
la23	18 (100.0%)	0 (0.0%)
la24	3 (17.0%)	0 (0.0%)
la25	0 (0.0%)	0 (0.0%)
la26	15 (83.0%)	0 (0.0%)
la27	8 (44.0%)	0 (0.0%)
la28	11 (61.0%)	0 (0.0%)
la29	2 (11.0%)	0 (0.0%)
la30	18 (100.0%)	0 (0.0%)
la31	15 (83.0%)	0 (0.0%)

	Mistral	CPO
la32	17 (94.0%)	0 (0.0%)
la33	16 (89.0%)	0 (0.0%)
la34	16 (89.0%)	0 (0.0%)
la35	18 (100.0%)	0 (0.0%)
la36	1 (6.0%)	0 (0.0%)
la37	4 (22.0%)	0 (0.0%)
la38	0 (0.0%)	0 (0.0%)
la39	3 (17.0%)	0 (0.0%)
la40	0 (0.0%)	0 (0.0%)
orb01	0 (0.0%)	0 (0.0%)
orb02	0 (0.0%)	0 (0.0%)
orb03	0 (0.0%)	0 (0.0%)
orb04	0 (0.0%)	0 (0.0%)
orb05	0 (0.0%)	0 (0.0%)
orb06	0 (0.0%)	0 (0.0%)
orb07	0 (0.0%)	0 (0.0%)
orb08	7 (39.0%)	0 (0.0%)
orb09	0 (0.0%)	0 (0.0%)
orb10	1 (6.0%)	0 (0.0%)
swv01	0 (0.0%)	0 (0.0%)
swv02	4 (22.0%)	0 (0.0%)
swv03	0 (0.0%)	0 (0.0%)
swv04	0 (0.0%)	0 (0.0%)
swv05	0 (0.0%)	0 (0.0%)
swv06	0 (0.0%)	0 (0.0%)
swv07	0 (0.0%)	0 (0.0%)
swv08	0 (0.0%)	0 (0.0%)
swv09	0 (0.0%)	0 (0.0%)
swv10	0 (0.0%)	0 (0.0%)
swv11	1 (6.0%)	0 (0.0%)
swv12	0 (0.0%)	0 (0.0%)
swv13	0 (0.0%)	0 (0.0%)
swv14	2 (11.0%)	0 (0.0%)
swv15	0 (0.0%)	0 (0.0%)
swv16	17 (94.0%)	0 (0.0%)
swv17	18 (100.0%)	0 (0.0%)
swv18	18 (100.0%)	0 (0.0%)
swv19	14 (78.0%)	0 (0.0%)
swv20	18 (100.0%)	0 (0.0%)
Total	582 (41.5%)	74 (5.3%)

Figure 3 Number of instances where an optimal solution was found