Using Constraint Programming for the Urban Transit Crew Rescheduling Problem

Xavier Lorca, Charles Prud'Homme, Aurélien Questel, Benoît Rottembourg

▶ **To cite this version:**

Xavier Lorca, Charles Prud'Homme, Aurélien Questel, Benoît Rottembourg. Using Constraint Programming for the Urban Transit Crew Rescheduling Problem. Principles and Practice of Constraint Programming, Sep 2016, Toulouse, France. pp.636 - 649, 10.1007/978-3-319-44953-1_40 . hal-01436288

HAL Id: hal-01436288
https://hal.science/hal-01436288

Submitted on 16 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using Constraint Programming for the Urban Transit Crew Rescheduling Problem

Xavier Lorca, Charles Prud'homme, Aurélien Questel, and Benoît Rottembourg

TASC - École des Mines de Nantes, Université de Nantes, INRIA, LINA UMR 6241
{firstname.lastname}@mines-nantes.fr
EURODECISION Versailles
{firstname.lastname}@eurodecision.com

Abstract. Scheduling urban and trans-urban transportation is an important issue for industrial societies. The Urban Transit Crew Scheduling Problem is one of the most important optimization problem related to this issue. It mainly relies on scheduling bus drivers' workday respecting both collective agreements and the bus schedule needs. If this problem has been intensively studied from a tactical point of view, its operational aspect has been neglected while the problem becomes more and more complex and more and more prone to disruptions. In this way, this paper presents how the constraint programming technologies are able to recover the tactical plans at the operational level in order to efficiently help in answering regulation needs after disruptions.

1 Context and opportunities

Scheduling urban and trans-urban transportation is an important issue for industrial societies. Several aspects are considered by territorial collectivities and transportation operators: human and material resource for, environmental and social constraint enforcement, user need requirements. Basically, there exists two central problems: transit scheduling (vehicles - buses, trains, tramways - planning on routes) and driver duty planning (assigning crew to those routes). These problems become more and more complex (regulation, network expansion) and more and more prone to *disruptions* (city events, accidents, resource failures) in the operational phase.

This is the context in which the Urban Transit Crew Scheduling Problem (UTCSP) has been introduced [5]. In our proposal, we have to schedule bus drivers' workdays according to several constraints mainly related to: (1) collective agreements, *e.g. breaking rules* (basically, how long a bus driver can work before a break) ; (2) the bus schedule itself, *e.g. chaining rules* (geographic position, schedule compatibility between two tasks, etc.). From a tactical point of view, the UTCSP has been thoroughly studied both at the academic and industrial levels. Primarily, the technologies derived from mathematical programming (from integer linear programming to column generation and dynamic programming) dominate the literature and offer satisfying results. The problem is mainly solved using a set covering approach that represents bus schedule as a set of tasks, linked together by chaining rules and respecting the breaking rules. From an operational point of view, these technologies become useless due to their time consumption. Such a pitfall leads the operators to manually repair the tactical solutions after a

disruption (at the operational level), and to consequently derive the applied schedule quality.

Focusing on the operational point of view of the UTCSP, *a.k.a.* Urban Transit Crew Rescheduling Problem (UTCRP), this paper presents how the Constraint Programming (CP) technologies are able to recover the tactical plans at the operational level in order to efficiently help answer regulation needs after disruptions. Precisely, it is shown how a constraint model of the UTCSP can be simply modified to address the UTCRP, its operational reformulation.

The paper is composed of six parts. The present one has introduced the context of the UTCSP and has motivated the opportunities for the CP technologies to tackle the rescheduling problem, namely the UTCRP. Next, Section 2 is dedicated to the related works and the operational context of the UTCSP. Section 3 is the main section of the paper. A CP model for the UTCSP is first presented. Next, it is shown how simple modifications of the UTCSP lead to a model for the UTCRP. Then, Section 4 introduces a common search strategy for UTCSP and its operational version. Section 5 reports the empirical evaluations of both UTCSP and UTCRP on industrial instances. Finally, Section 6 concludes.

2 Related Works and Operational context

The most widely used approach for this problem is related to a set partitioning problem [2]. It is mainly solved by a Branch-and-Bound algorithm where the lower bound is computed through a column generation procedure [4]. In this case, the subproblem corresponds to an Elementary Shortest Path Problem with Resource Constraints (ESPPRC). Most of the time, it is solved using dynamic programming [6, 10]. However, according to the number and the nature of the resource constraints, generating a pool of column during a preprocessing may sometimes be more efficient [11]. The resulting Branch-and-Price algorithm is designed to be an exact model which, nevertheless, may fail to optimally solve very large problems (with more than thousands tasks) [3, 5, 20]. To bypass this issue, a common approach consists in truncating the search tree [8], the lower bound quality is ensured then to be the nearest possible from optimal solutions. Even if many meta-heuristic algorithms have been proposed for the UTCSP [20, 7, 12, 19], the most efficient industrial softwares such as Austrics,[1] Hastus,[2] GoalDriver[3] or LP-EasyDriver [4] are based on a truncated version of the exact Branch-and-Price algorithm.

Constraint programming attempt. The UTCSP has not been many studied by the CP community. The most related work is [21]. It introduced a pure CP model for a variant of the UTCSP, as well as a hybrid approach composing a column generation algorithm with a CP model dedicated to generate the columns. Nevertheless, the proposed pure CP approach is quite poor because it does not embed global constraints, like REGULAR,

[1] http://www.trapezegroup.com
[2] http://www.giro.ca
[3] http://www.goalsystems.com
[4] http://www.eurodecision.fr

to express the regulation needs. Moreover, the search strategy is limited to the *first-fail* principle which is clearly inappropriate. Consequently, not more than 30 tasks can be scheduled to optimality and feasible solutions can be provided with only at most 125 tasks. However, the main contribution is related to the hybridization. Thus, it is shown that the hybrid approach (combining column generation and CP) can be efficient up to 150 tasks.

Operational context. To the best of our knowledge, there is no literature dealing with the rescheduling of the UTCSP. So, we present an industrial point of view which motivates our proposal. Disruptions can occur a few days before the buses run. Typically, some special event is programmed in the city, like a football match or local roadworks and the bus schedules must be adapted "accordingly". Without loss of generality, tasks to be performed by the buses can either be added, deleted or their duration modified. In some cases buses have connections with trains at the coach station, and a slight change of the train timetable (imposed by French Railways for instance) can have an impact on the bus task at this place in the network. From an operational point of view, a compromise must be found by the planning team: on the one hand, building a new cost-efficient schedule covering the modified tasks and, on the other hand, building a schedule not "too different" from the regular daily schedules. For bus drivers, new workdays might have an impact on either security or comfort, for instance when the new workday ends later than usual.

Today, tools are mainly focussed on finding optimal solutions and tends to produce - when tasks are altered - bran-new schedules that "destroy" the daily ones. In practice, human operators adopt a workaround strategy and manually fix large parts of the schedule, asking the solver tool to optimize only subparts of the schedule. This approach clearly avoids too much perturbation but consequently offer suboptimal solutions. It is also a time consuming effort. The UTCRP consists then in managing a good balance between cost optimality and schedule updates, with dedicated metric and constraints on top on the classical ones. On the converse to pure tactical crew scheduling tools, the computational time of rescheduling must be short, as various trials and errors can have to be experimented by the planning team which is evaluating the impact of what-if scenarios.

3 Constraint programming model

In the UTCSP, tasks have to be assigned to bus drivers. Formally, given n tasks that are to be assigned to at most m bus drivers, the objective is to find a full assignment that minimizes the cost and satisfies breaking rules and chaining rules. A bus driver is mainly characterized by a unique identifier, a skill level, among *novice* and *expert*, and a workday duration and a hourly cost induced by its skill. A novice can only execute low level skill tasks while an expert can execute any type of tasks and can work longer but at a higher hourly cost. The maximal number of novices, $nbNovices$ and experts, $nbExperts$, needed to trivially satisfy the problem is determined from the input. A same bus driver can perform many tasks, but one task is only processed by a single bus driver. Each task is defined by its fixed beginning time, B_i, its fixed duration, D_i, its fixed

end time, E_i such that, $E_i = B_i + D_i$, $\forall i \in 1..n$. The initially unknown bus driver performing the task is denoted A_i. In addition, a task requires exactly one skill, K_i: either high level skill, only experts can do it, or low level skill, anyone can do it.

A first constraint is that tasks assigned to the same bus driver should not overlap in time. A second constraint deals with the allocation of breaks to each bus driver. This is done by stating a SHIFT constraint, which we now describe.

Definition 1 (Shift). *A* shift *is a maximum sequence of tasks assigned to a same bus driver such that the time gap between any two consecutive tasks is shorter than a given threshold* $minBreak$.

Two consecutive shifts of a bus driver are separated by a *break* of minimum duration $minBreak$ and define its workday. Two consecutive tasks of a shift are separated by a gap shorter than $minBreak$. The *span* of a shift is the difference between the end time of its last task and the beginning time of its first task. It is bounded by a given threshold $maxSpan$.

3.1 Modeling the SHIFT constraint

Concisely expressing constraints like SHIFT is hard and has been recently studied. In [1], a possible model is presented to manage the SHIFT problem based on the REGULAR [14] and GLOBAL CARDINALITY [17] constraints. We do not report such a model in this paper because of its time generation and memory consumption. Another interesting model, also introduced in [1], addresses most of the pitfalls of the previous one: a STABLEKEYSORT-based model. In such a model, a *decomposition* of the SHIFT constraint is expressed as a conjunction of a STABLEKEYSORT constraint and simple arithmetical and logical constraints. The STABLEKEYSORT(L, P, S, k) constraint is declared with:

- $L = \langle A_i, B_i, D_i, E_i \rangle \mid i \in 1..n]$: a list of task attribute tuples,
- P: an optional permutation list (not required here),
- $S = \langle A'_i, B'_i, D'_i, E'_i \rangle \mid i \in 1..n]$: a stable and non decreasing rearrangement of L,
- $k = 2$: number of first positions to consider in the tuples.

Doing so, the STABLEKEYSORT(L, P, S, k) provides a *view* of tasks of L in which tasks are sorted by workdays. Hence, it eases the expression of the required constraints which can be directly expressed on sorted variables, while their expression could be more tedious otherwise.

Decomposition 1 depicts how the SHIFT constraint is expressed using a STABLEKEYSORT constraint in the state-of-the-art. Constraint (1) ensures tasks integrity. Constraint (2) maintains the rearrangement of task attribute tuples, here only A_i and B_i are considered to sort tuples. Constraint (3) ensures that two consecutive tasks either belong to the different workdays or are chronologically ordered in the same workday and thus do not overlap in time. Constraints (4) and constraint (5) introduce auxiliary $0..1$ variables: Y_i indicates whether two consecutive tasks $i-1$ and i are in the same workday, X_i indicates if two tasks $i-1$ and i are in the same shift. Finally, a last set of auxiliary $1..maxSpan$ variables R_i are needed in constraint (6), they compute the shift length up to the end of task i.

$$\text{SHIFT}([\langle A_i, B_i, D_i, E_i\rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i, Y_i\rangle \mid i \in 1..n], minBreak, maxSpan) \Leftrightarrow (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6)$$

$$E_i = B_i + D_i,\ \forall i \in 1..n \tag{1}$$

$$\text{STABLEKEYSORT}([\langle A_i, B_i, D_i, E_i\rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i\rangle \mid i \in 1..n], 2) \tag{2}$$

$$A'_{i-1} < A'_i \ \vee\ E'_{i-1} \leq B'_i,\ \forall i \in 2..n \tag{3}$$

$$Y_i = 1 \Leftrightarrow \begin{cases} \textbf{false} & \text{if } i = 1 \\ A'_i = A'_{i-1} & \text{if } i \in 2..n \end{cases} \tag{4}$$

$$X_i = 1 \Leftrightarrow \begin{cases} \textbf{false} & \text{if } i = 1 \\ Y_i \ \wedge\ B'_i - E'_{i-1} < minBreak & \text{if } i \in 2..n \end{cases} \tag{5}$$

$$R_i = \begin{cases} D'_i & \text{if } i = 1 \\ D'_i + X_i \cdot (R_{i-1} + B'_i - E'_{i-1}) & \text{if } i \in 2..n \end{cases} \tag{6}$$

where

$$R_i \in 1..maxSpan,\ \forall i \in 1..n$$

Decomposition 1: Decomposition of the SHIFT constraint [1].

3.2 A constraint-based model for the UTCSP

The targeted problem is based on a central SHIFT constraint but also comes with the following additional variables and constraints, presented in Model 1. For modeling purpose, bus drivers whom unique identifier is in $1..nbNovices$ are novices, those with greater identifier, up to $nbNovices + nbExperts$, are experts. Hence, constraint (8) introduces the auxiliary $0..1$ variables K_i which indicate whether the bus driver performing the task i is an expert. P_i, W_i, C_i denote respectively the period between two consecutive tasks $i-1$ and i, stated by constraint (9), the duration of a shift including task i, stated by constraint (11), and the cost of a bus driver's workday, stated by constraint (12). Note that the workday duration is bounded thanks to that last constraint. It is expressed with a TABLE [9] constraint wherein possible combinations of skill and workday and costs are listed. Constraint (10) ensures that, in a shift, the period between two tasks $i-1$ and i is less than or equal to the $maxSpan$. Based on an *a priori* analysis of the tasks network, ALLDIFFERENT [16] constraints (13) make sure that tasks belonging to the same clique are performed by different bus drivers.

Finally, the model is improved by considering symmetries and their related symmetry breaking constraints, depicted by Model 2. First, we introduce M_i^R and M_i^E which maintain, for each skill, the list of identifier of used bus drivers (constraints (14) and (15)), they also help to count the number of bus drivers per skill. Then, constraints (16), (17), (18) and (19) break symmetries.

3.3 A constraint-based model for the UTCRP

In order to stay close to the operational context of the bus networks, we have built instances for the UTCRP based on disruptions of UTCSP instances. Indeed, a daily schedule is considered as known, and the tasks' disruptions occur locally. So, the disruptions can be easily simulated from the UTCSP instances. We have to keep in mind that these disruptions can either be:

Minimize

$$\sum_{i=1}^{n} C_i \tag{7}$$

subject to

$$\textsc{Shift}([\langle A_i, B_i, D_i, E_i\rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i, Y_i\rangle \mid i \in 1..n], minBreak, maxSpan)$$

$$K_i = 1 \Leftrightarrow A'_i > nbNovices,\ \forall i \in 1..n \tag{8}$$

$$P_i = \begin{cases} 0 & \text{if } i = 1 \\ B'_i - E'_{i-1} & \text{if } i \in 2..n \end{cases} \tag{9}$$

$$Y_i \Rightarrow P_i \leq maxSpan,\ \forall i \in 2..n \tag{10}$$

$$W_i = \begin{cases} D'_i & \text{if } i = 1 \\ D'_i + X_i \cdot (W_{i-1} + P_i) & \text{if } i \in 2..n \end{cases} \tag{11}$$

$$C_i = \begin{cases} (1 - Y_{i+1}) \cdot cost_{K_i,W_i} & \text{if } i \in 1..n-1 \\ cost_{K_i,W_i} & \text{if } i = n \end{cases} \tag{12}$$

Redundant constraint

$$\textsc{AllDifferent}([A_k \mid k \in clique(A)]) \tag{13}$$

where $\forall i \in 1..n$,

$$P_i \in -dayDuration..dayDuration,\ W_i \in 0..dayDuration,\ C_i \in 0..dayDuration \times costExpert,$$

Model 1: Formulation of the Urban Transit Crew Scheduling Problem.

- Creation of new tasks: the bus line has to serve the stadium station at 1 o'clock in the morning, due to a rock concert;
- Task deletion: a roadwork prevents the bus to use the road and stop at the station;
- Duration change: roadworks slow down the traffic;
- Start time change: a train timetable has been modified, and the connection forces a change of the bus departure time.

Consequently, UTCRP aims at producing new schedules which: a) cover all the tasks (previous and new); b) do not differ too much in terms of cost from the initial daily schedule; and c) ensure that the workday modifications are minimal for the bus drivers. These schedules have to be produced within a few seconds of computational time so that the planning team can test different scenarios and choose the most convenient one.

From a constraint programming point of view, Model 3 presents how to turn the UTCSP model, depicted in Section 3.2, into UTCRP. Given I_s, an assignment of n tasks to m bus drivers, let us denote $C^\star$ its cost. Consider that some of the tasks have been disrupted and that I_s has to be repaired. First, the previous objective function, equation (7) in Model 1, is turned into the hard constraint (21): new solutions have to be at most $\epsilon\%$ above $C^\star$. Then, for each set of tasks performed by the same bus driver in I_s, the number of bus drivers needed to perform the same tasks anew is maintained by constraint (22). Finally, the objective function (20) aims at computing solutions similar to I_s in term of unmodified workdays.

$$M_i^R = \begin{cases} A'_i & \text{if } K_i = 0 \\ 0 & \text{if } K_i = 1 \wedge i = 1 \\ M_{i-1}^R & \text{if } K_i = 1 \wedge i \in 2..n \end{cases} \tag{14}$$

$$M_i^E = \begin{cases} A'_i & \text{if } K_i = 1 \\ nbNovices & \text{if } K_i = 0 \wedge i = 1 \\ M_{i-1}^E & \text{if } K_i = 0 \wedge i \in 2..n \end{cases} \tag{15}$$

$$M_{i-1}^R \leq M_i^R,\ \forall i \in 2..n \tag{16}$$

$$M_{i-1}^E \leq M_i^E,\ \forall i \in 2..n \tag{17}$$

$$\textsc{IntValuePrecedeChain}(A, [j \mid j \in 1..nbNovices]) \tag{18}$$

$$\textsc{IntValuePrecedeChain}(A, [j \mid j \in nbNovices..nbNovices + nbExperts]) \tag{19}$$

where $\forall i \in 1..n$,

$$M_i^R \in 1..nbNovices,\ M_i^E \in nbNovices..nbNovices + nbExperts$$

Model 2: Improving Model 1 with symmetry breaking constraints.

Minimize

$$\sum_{\ell=1}^{m} N_\ell \tag{20}$$

subject to

$$\sum_{i=1}^{n} C_i < (C^\star \cdot \epsilon)/100 \tag{21}$$

$$\textsc{NValues}([A_\ell \mid \ell \in workday(A)], =, N_\ell) \tag{22}$$

where $\forall \ell \in 1..m$,

$$N_\ell \in 0..n$$

Model 3: Modifications to bring to the CP model of UTCSP to turn it into UTCRP.

4 Search strategy

A constructive search strategy is defined to dive to a first solution quickly without failure. The A_i variables are selected in lexicographic order. The bus driver that performs a task A_i is computed as follow. Bus drivers already performing at least one task are considered first. Those whose workday is not directly compatible with the task to assign (*w.r.t.* either break rules or chaining rules cannot be satisfied) are ruled out. Remaining ones are then tried sequentially. Some more tries are finally considered, allowing the addition of at most one bus driver per type of skill required.

Next solutions are obtained by plugging Large Neighborhood Search (LNS) [18] in, a straightforward two-phase local search-like approach. It partially *relaxes* a given solution and tries to *repair* it. Given an input solution, the relaxation phase builds a *partial solution*: some variables are selected to be relaxed to their initial domain, while the other ones are assigned to their value in the solution. The reparation phase tries to extend the partial solution to a complete one that improves the objective function.

The efficiency of LNS lies in the way variables are selected to be relaxed. In our case, it tries to find a better solution by locally rearranging bus drivers' workday. So,

the workdays are first extracted from the A variables in a solution. Then, up to $\theta \in [\![2, 4]\!]$ workdays are randomly selected to be rearranged in such way that they overlap in time at least one of the other selected one. The corresponding A variables are relaxed, the other ones are assigned to the same bus driver as declared in the solution. To consolidate even more the partial solution, bus drivers' identifier corresponding of fixed A variables are removed from relaxed A variables' domain. The process is completed with a fast restart strategy [13], which limits the reparation phase to $2n$ failures and avoids spending too much time in hard-to-repair partial solutions.

The same search strategy is applied to both the UTCSP and the UTCRP. The strategy was initially designed to produce few dense workdays, with a local reasoning. When the objective changes to repair assignments, the model is constrained enough to guide the process towards workdays similar to the initial ones.

5 Practical Experiments

For an empirical evaluation, we instantiate the UTCSP model of Section 3.2. The SHIFT constraint holds for a 15-minute $minBreak$ and 2-hour $maxSpan$. A novice cannot work more than 8 hours and its hourly cost is fixed to 10. An expert cannot work more than 9 hours, and its hourly cost is fixed to 17. We consider here the following additional rule: any working bus driver has a minimal 6-hour pay, even if its workday lasts for less than 6 hours. This constraint is directly encoded into constraint (12). Finally, the UTCRP model depicted in Section 3.3 also instantiates ϵ, the distance to $C^\star$, to 10.

There are two sets of instances. A first set aims at comparing the constraint-based approach with the state-of-the-art one on the UTCSP. It is composed of real-world problem instances involving up to 3,200 tasks. A second set aims at evaluating how repairing a disrupted assignment is made easy with a constraint-based approach using the UTCRP (Model 3). First of all, the best solution found by EURODECISION for the instance with 800 tasks is selected, it is composed of 160 workdays. Then, this instance is disrupted applying the following process: a task j is randomly selected to be removed. Starting from j, within a range of $\beta \in \{30, 60, 90, 120\}$ minutes before B_j and after E_j, tasks on a path to j, that is *w.r.t.* breaking rules and chaining rules, are removed. Tasks that overlap the range in time are reduced (either B_i or E_i is modified). The selection-and-removing phase is repeated $\alpha \in [\![1..5]\!]$ times. This results in a set of 20 instances to be repaired with up to 25% disrupted tasks, which corresponds to our real life context.

Protocol. The experiments were run on a Mac Pro with 8-core Intel Xeon E5 at 3 GHz under MacOS 10.11.3 and Java 1.8.0_25. Each instance of the first set was run with a 2-hour limit on its own core, and each instance of the second set was run with a 1-minute limit on its own core for CP approach. LP-EasyDriver was evaluated in a 2-hour limit for both sets. All of them were run with up to 4 GB of memory. The tools used are: Choco [15] for the constraint programming part, while EURODECISION provides LP-Taskplanner, a generic framework for ESPPRC based column generation models, relying on CPLEX 12.6.2. LP-Taskplanner is embedded in LP-EasyDriver: the latter handles every business aspects while the former deals with the optimization parts. In the following, this tool will be referred to as LP-EasyDriver.

5.1 Solving the UTCSP

In this section, we compare the constraint-based approach for the UTCSP with the one being used by EURODECISION and introduced in Section 2. The results are given in Table 1 which is divided into four parts. The first part indicates the instance size n. The second part reports information related to the constraint-based approach: the time to get the first solution (**time**, in seconds), its cost (**first**) and the best cost obtained in 120 minutes (**best@120**). The third part is about LP-EasyDriver: the time (**time**, in seconds) to compute the lower bound (**LB**) and a first upper bound (**UB**) then the best upper bound in 120 minutes (**UB@120**). Finally, the last part reports, when possible, the ratio

$$g(a, b) = \frac{a}{b} \times 100$$

where, here, $a = (\textit{best@120} - \textit{UB@120})$ and $b = \textit{UB@120}$. In addition, "*N/A*" denotes that the information is not available or applicable, "*OOT*" denotes that a particular approach runs out of time.

n	Constraint-based approach			LP-EasyDriver					**Gap**
	time	**first**	**best@120**	**time**	**LB**	**time**	**UB**	**UB@120**	
200	0.97	5005.00	4352.75	9	4069.81	10	4150.25	4073.75	6.85
400	1.71	9131.75	8318.50	134	7654.78	140	8790.75	7677	8.36
600	2.65	13566.00	12218.25	499	10845.6	4152	11328.5	11014	10.93
800	3.80	17479.25	16375.00	1265	14472.8	23128	14690.2	*N/A*	*N/A*
1000	5.51	20857.75	20211.25	2593	17868	*OOT*		*N/A*	*N/A*
1400	9.42	29210.75	28031.00	10245	24214.2	*OOT*		*N/A*	*N/A*
1800	15.22	37394.50	36431.00	33698	31578.2	*OOT*		*N/A*	*N/A*
2200	22.42	44849.50	43856.50	*OOT*		*OOT*		*N/A*	*N/A*
2600	31.94	52885.75	52011.25	*OOT*		*OOT*		*N/A*	*N/A*
3000	45.09	59138.50	58656.25	*OOT*		*OOT*		*N/A*	*N/A*
3200	52.60	65382.75	64584.75	*OOT*		*OOT*		*N/A*	*N/A*

Table 1. Empirical results on the UTCSP: time (**time**, in seconds) for finding the first solution (**first**) and the best solution found in 120 minutes (**best@120**), and for finding the root lower bound (**LB**), the first upper bound (**UB**), and the best upper bound in 120 minutes (**UB@120**). The ratio of best@120 to UB@120 is also reported (**Gap**, in %).

We observe that:

- The truncated Branch-and-Price approach fails at solving large problems. This observation confirms the state-of-the-art (Section 2). The bounds produced are very sharp considering a 2-hour time limit. Indeed, such a time limit is below from what is commonly allocated at the tactical level.
- The constraint-based approach is able to find a first solution for any problem size. Moreover, the ability of the model to scale up combined with a constructive search strategy makes possible to provide a first solution in a very short time. Nonetheless, the quality of the best solution found in the given time limit tends to decrease when the instance size n becomes larger. In the end, without any evaluation - neither propagation - of the lower bound, the approach fails at proving optimality.

At a tactical level, where the run time matters less than the quality of the bounds, and up to mid-size problems, the state-of-the-art approach is suitable. On any-size problems, the constraint-based approach is responsive, yet less accurate. At an operational level, when rescheduling tasks is needed, responsiveness becomes indispensable. This is the point we want to evaluate in Section 5.2. e

5.2 Solving the UTCRP

When a disrupt occurs, its extent is measurable thanks to three indicators: the number of removed or modified tasks (n'), the number of bus drivers whose workday is disrupted (m') and the total number of affected tasks, gathering all tasks of disrupted workdays (n''). In practice, when dealing with rescheduling, not all the tasks are set as input of the model depicted in Section 3.3. Indeed, only affected tasks are considered. The remaining workdays are immutable. Recall that the solution selected to be disrupted was made of $n = 800$ tasks and $m = 160$ bus drivers (or workdays).

We report the results of constraint-based approach for the UTCRP. The results are given in Table 2 which is composed of four parts. The first part indicates the disruption parameters β and α and the indicators of its extent n', m' and n''. The second part reports information related to the constraint-based approach: the time to get the first solution (**time**, in seconds) and **gap@first** that is the ratio $g(m-m'+m^1, m)$ where m^1 is the number of workdays covering the n'' tasks (recall that the unaffected workdays are kept immutable). This qualifies the distance from the first solution found to the initial one in terms of number of modified workdays needed to cover the affected tasks. The ratios based on the best solution found in ten seconds (**gap@10**) and 60 seconds (**gap@60**) relative to the initial solution are also reported. Then, the cost of the best solution found in 60 seconds is indicated (**cost@60**). For indication purpose, the results found with LP-EasyDriver when solving the problems from scratch, without any time limit, are reported in the third part (**best**). Finally, the ratio $g(\textit{cost@60} - \textit{best}, \textit{best})$ can be observed in the last part (**Gap**). In addition, "-" denotes that no better solution was found in the elapsed time.

Our observations are threefold:

- The constraint-based approach is responsive as expected. Its ability to quickly reschedule disrupted workdays does not depend on the extent of the disruption.

Inst. parameters					Constraint-based approach					LP-EasyDriver	Gap
β	α	n'	m'	n''	**time**	**gap@first**	**gap@10**	**gap@60**	**cost@60**	**best**	
	1	3	3	15	0.85	1.87	0.00	-	14489.50	14462.00	0.19
	2	7	7	39	0.98	4.37	0.62	-	14584.75	14482.50	0.71
30	3	11	10	63	1.07	15.62	2.50	0.62	14576.25	14440.00	0.94
	4	15	14	92	1.16	27.50	20.62	3.75	14597.50	14369.25	1.59
	5	18	17	114	1.25	35.62	31.87	4.37	14729.75	14336.50	2.74
	1	4	4	21	0.86	2.50	0.62	-	14591.50	14446.00	1.01
	2	9	9	41	0.93	4.37	0.62	-	14532.00	14416.00	0.80
60	3	15	14	72	1.06	10.62	3.12	0.62	14578.00	14373.50	1.42
	4	18	16	83	1.09	13.12	2.50	0.62	14497.25	14314.50	1.28
	5	22	18	99	1.11	18.75	11.87	0.62	14539.75	14345.00	1.36
	1	6	6	35	0.91	3.75	0.62	-	14576.25	14464.00	0.78
	2	13	12	76	1.07	13.75	2.50	1.25	14623.00	14466.75	1.08
90	3	16	14	84	1.09	16.25	5.62	1.87	14631.50	14424.00	1.44
	4	24	21	125	1.24	29.37	23.12	1.25	14669.25	14434.25	1.63
	5	32	28	168	1.37	42.50	41.25	2.50	14669.75	14294.00	2.63
	1	8	7	41	0.94	6.25	1.25	-	14563.50	14458.50	0.73
	2	18	15	103	1.18	29.37	19.37	0.62	14571.25	14423.00	1.03
120	3	27	20	134	1.31	38.12	31.87	0.62	14551.00	14334.25	1.51
	4	37	28	185	1.39	55.62	53.12	3.75	14743.25	14319.00	2.96
	5	44	33	209	1.45	63.75	61.25	3.75	14722.50	14259.50	3.25

Table 2. Empirical results on the UTCRP: range of the disruption (β) along a path involving a randomly selected task, number of randomly selected tasks (α), number of directly affected tasks (n') and bus drivers (m') and total number of affected tasks (n''), time (**time**, in seconds) for finding the first solution, the ratio of the number of workdays needed in, respectively, the first solution (**gap@first**, in %), the solution found in 10 seconds to (**gap@10**, in %) and the solution found in 60 seconds (**gap@60**, in %) to the initial number of workdays required, the cost of the solution found in 60 seconds (**cost@60**), and the ratio of cost@60 to the best solution (**best**) computed from scratch with the LP-EasyDriver (**Gap**, in %).

Nevertheless, the quality of the reparation is related to the total number of the affected tasks n''.

- The search strategy is able to compute good quality solutions in term of number of workdays. When less than 83 tasks are affected by the disruption, ten seconds are enough to provide solutions at most 5% above the number of workdays needed before the disruption. When more tasks are affected, sixty seconds are always enough to find solution at most 5% above the initially required number of workdays.
- The search strategy is able to compute good quality solution in terms of cost. The best solutions found with CP are very close to the ones computed from scratch by LP-EasyDriver. Nevertheless, the EURODECISION approach fails at maintaining the workdays which are not impacted by disruptions. Consequently, such approach does not fit the needs of the planning teams.

6 Conclusion and Further Works

In the context of scheduling bus drivers' workdays, this paper first described a constraint-based model for the UTCSP. We empirically confirmed that this CP model scaled up to 3,200 tasks and was able to quickly build good quality solutions. Nevertheless, due to a lack of any lower bound integration, it struggled to improve them enough to reach and prove optimality. Second, we highlighted CP flexibility by turning the UTCSP into

its operational counterpart, the UTCRP. The ability of the constraint-based approach to quickly reschedule workdays has been assessed on randomly disrupted instances. Indeed, the closeness of the new scheduling to the initial one in terms of both workdays and cost bears out that CP is a serious runner for that problem.

The responsiveness and the good quality of the solutions found by the CP approach for UTCRP offer new perspectives and advocates for field experiments. EURODECISION will launch pilots with groups of customers in order to measure the ease of use of repaired solutions together with the real life metrics used by the planning teams. Our goal is to reduce the hassle imposed to the planners by the disruptions without loosing too much cost compared to optimality.

Acknowledgements

The authors are supported by the French common-laboratory grant "*TransOp*" involving the TASC team and EURODECISION.

References

1. Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, Xavier Lorca, Justin Pearson, Thierry Petit, and Charles Prud'Homme. A modelling pearl with sortedness constraints. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *GCAI 2015. Global Conference on Artificial Intelligence*, volume 36 of *EPiC Series in Computing*, pages 27–41. EasyChair, 2015.
2. Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371, 2000.
3. Shijun Chen and Yindong Shen. An improved column generation algorithm for crew scheduling problems. *Journal of Information and Computational Science*, 10(1):175–183, 2013.
4. Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.
5. Martin Desrochers and Franois Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, 1989.
6. Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.
7. Paul Forsyth and Anthony Wren. An ant system for bus driver scheduling1. 1997.
8. Birger Franck, Klaus Neumann, and Christoph Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23(3):297–324, 2001.
9. Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Data structures for generalised arc consistency for extensional constraints. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 191–197. AAAI Press, 2007.
10. Stefan Irnich, Guy Desaulniers, et al. Shortest path problems with resource constraints. *Column generation*, 6730:33–65, 2005.
11. Éric Jacquet-Lagrèzel. Horaires de chauffeurs de bus. *Gestion de production et ressources humaines: méthodes de planification dans les systèmes productifs*, page 287, 2005.

12. Helena R Lourenço, José P Paixão, and Rita Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation science*, 35(3):331–343, 2001.
13. Laurent Perron. Fast restart policies and large neighborhood search. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2003.
14. Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.
15. Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2015.
16. Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 362–367. AAAI Press / The MIT Press, 1994.
17. Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 209–215. AAAI Press / The MIT Press, 1996.
18. Paul Shaw. *Principles and Practice of Constraint Programming — CP98: 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings*, chapter Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, pages 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
19. Yindong Shen and Raymond SK Kwan. *Tabu search for driver scheduling*. Springer, 2001.
20. Gustavo Peixoto Silva and Allexandre Fortes da Silva Reis. A study of different metaheuristics to solve the urban transit crew scheduling problem. *Journal of Transport Literature*, 8(4):227–251, 2014.
21. Tallys H. Yunes, Arnaldo Vieira Moura, and Cid C. de Souza. A hybrid approach for solving large scale crew scheduling problems. In *Practical Aspects of Declarative Languages, Second International Workshop, PADL 2000, Boston, MA, USA, January 2000, Proceedings*, volume 1753 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2000.