A Time-Dependent No-Overlap Constraint: Application to Urban Delivery Problems

Penélope Aguiar Melgarejo, Philippe Laborie, Christine Solnon

A Time-Dependent No-Overlap Constraint: Application to Urban Delivery Problems

Penélope Aguiar Melgarejo[1,2,3], Philippe Laborie[3], Christine Solnon[1,2]

[1] Université de Lyon, CNRS
[2]INSA-Lyon, LIRIS, UMR5205, F-69621, France
christine.solnon@insa-lyon.fr
[3] France Lab, IBM, Gentilly, France
{*penelopeam,laborie*}*@fr.ibm.com*

Abstract. The Time-Dependent Traveling Salesman Problem (TDTSP) is the extended version of the TSP where arc costs depend on the time when the arc is traveled. When we consider urban deliveries, travel times vary considerably during the day and optimizing a delivery tour comes down to solving an instance of the TDTSP. In this paper we propose a set of benchmarks for the TDTSP based on real traffic data and show the interest of handling time dependency in the problem. We then present a new global constraint (an extension of no-overlap) that integrates time-dependent transition times and show that this new constraint outperforms the classical CP approach.

1 Introduction

When we consider real world optimization problems, time is usually an important dimension to take into account. This is particularly the case for Delivery Problems for which time is typically present in different forms: travel times between consecutive deliveries; time windows where deliveries are allowed; precedence constraints. We are interested in the Time-Dependent Traveling Salesman Problem (TDTSP), an extended version of the Traveling Salesman Problem (TSP) where the travel time between deliveries (visits) depends on the date of the travel. The TDTSP is at the core of many real-world scheduling problems such as urban delivery problems, for example, since traffic conditions in urban areas usually vary a lot during the day. In these real-world problems, there are frequently additional constraints such as time-windows or precedence constraints.

Since the availability of extensive real-world data in this area is quite recent, the TDTSP has not been much studied in the literature and Constraint Programming (CP) approaches are even rarer. One reason for this is that CP is usually less efficient than Integer Linear Programming or Meta-heuristics (Local Search, Evolutionary Algorithms, Ant Colonies) for pure (non time-dependent) vehicle routing problems. On the other hand, Constraint-Based Scheduling [4], that is the application of CP to scheduling problems, is one of the biggest industrial success of CP and has shown that CP technologies can be very efficient for

solving temporal problems. A variety of specialized variable types (interval variables, sequence variables) and related global constraints and search algorithms have been developed until recently [21, 22, 23] to improve the expressiveness and efficiency of CP-based models involving temporal domains.

In this paper, we start by giving a general definition of the TDTSP in Section 2. Then, we introduce a new benchmark for this problem in Section 3. This benchmark has been generated from real-world traffic data, coming from the city of Lyon, and we study the interest of handling time-dependent data on this benchmark. Section 4, describes related work. In Section 5, we introduce a basic CP model for the TDTSP and show some of its limitations. In Section 6, we describe a new global constraint (called TDnoOverlap) for efficiently tackling time-dependent data while ensuring that visits are not overlapping. This global constraint has been implemented on top of the CP Optimizer engine of IBM ILOG CPLEX Optimization Studio, and we experimentally evaluate it on our new benchmark in Section 7.

2 Definition of the TDTSP

The real world problem we address is the problem of scheduling a sequence of deliveries in a urban zone. The city of Lyon and other partners started a project called Optimod [1] with the goal to leverage city data to improve urban mobility. Traffic predictions are made from historic data and these predictions can be used to optimize moves in the city. In our case, we optimize delivery tours.

The theoretical problem involved is the TDTSP, an extension of the TSP where arc costs depend on the time when the arc is traveled.

In the TSP we are given a list of locations and the distances between every two of them. We are asked to find the tour minimizing the total traveled distance while visiting every location exactly once and coming back to the point of departure (depot).

In some cases though we are interested in minimizing the total travel time instead of distance or we simply need to schedule interventions or deliveries in certain time-windows predefined by the client. To do so we need to know the travel times between consecutive deliveries. To ensure a certain precision, since in urban zones travel times usually vary a lot during the day, we need time-dependent travel times. In order to take this variation into account we must know at which time the travel between two addresses starts so that we can take the time-dependent travel times into account.

We formally define the concepts of path, travel time function and timed-path in a graph and give a general formal definition of the TDTSP.

Definition 1 (Path). *A path $P = (v_1, ..., v_k)$ in a graph $G = (V, A)$ is a sequence of vertices such that $(v_i, v_{i+1}) \in A, \forall i \in \{1, ..., k-1\}, k \geq 2$.*

Definition 2 (Travel Time Function). *A travel time function $f : A \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is a function such that for a given arc $(v_i, v_j) \in A$, $f(v_i, v_j, t)$ is the travel time from v_i to v_j when leaving v_i at time t.*

In the case of deliveries and specially of interventions, the duration of visits is generally not the same so we consider that each visit v_i is associated with a given duration $d(v_i)$. The notion of Timed-Path extends the notion of path in the context of TDTSP.

Definition 3 (Timed-Path). *Given a graph $G = (V, A)$, a starting time $\tau \in \mathbb{R}^+$, a travel time function $f : A \times \mathbb{R}^+ \to \mathbb{R}^+$ and a duration function $d : V \to \mathbb{R}^+$, a timed-path $P_{\tau,f}$ in G is a path $(v_1, \ldots, v_k)$ such that each vertex v_i has an associated start time $t(v_i, P_{\tau,f})$, corresponding to the time of arrival at v_i. Furthermore start times must respect:*

$$t(v_1, P_{\tau,f}) \geq \tau$$

$$t(v_{i+1}, P_{\tau,f}) \geq t(v_i, P_{\tau,f}) + d(v_i) + f(v_i, v_{i+1}, t(v_i, P_{\tau,f}) + d(v_i)), \forall i \in \{1, ..., k\}$$

Definition 4 (TDTSP). *Given a graph $G = (V, A)$, a depot $s \in V$, a starting time $\tau \in \mathbb{R}^+$ and a travel time function $f : A \times \mathbb{R}^+ \to \mathbb{R}^+$, the Time-Dependent Traveling Salesman Problem is the problem of finding the timed-path $P_{\tau,f} = (v_1, \ldots, v_k)$ which starts from the depot ($v_1 = s$) and visits each vertex exactly once ($\{v_1, \ldots, v_k\} = V$), and such that the returning time to the depot, $t(v_k, P_{\tau,f}) + d(v_k) + f(v_k, s, t(v_k, P_{\tau,f}) + d(v_k))$, is minimal.*

We could consider minimizing different objective functions but for our application's purpose we consider that minimizing the end time is a good objective.

3 A new benchmark for the TDTSP

In the context of the Optimod project we had access to real traffic data measured from 630 sensors installed in the main axes of Lyon for 6 years. Those sensors measure vehicle's speed and estimations are made for the neighboring streets where there are no sensors. Given this data, a predictive model has been built, which gives predicted speeds on every street section, by 6-minute time steps.

We propose a benchmark generated using this model. The benchmark is built from 255 addresses randomly chosen from a list of delivery tours of transporters from Lyon, we can see their distribution in the city on Fig. 1.

Since the predictive model considers 6-minute time steps we produced a (stepwise) travel time function giving predicted travel times between every two addresses in the list for every 6-minute time step from 6h00 to 12h30 so that we have $m = 65$ time steps.

Travel times are computed using a time-dependent version of Dijkstra's point-to-point shortest path algorithm for every time step [27]. One limitation of our approach though is that we do not take into consideration the time spent in vertices in the path, where vertices are junctions between two or more street sections. We know from experience that the time it takes to traverse a crossroad or to turn left, for example, is an important factor in the augmentation of travel times during rush hours.

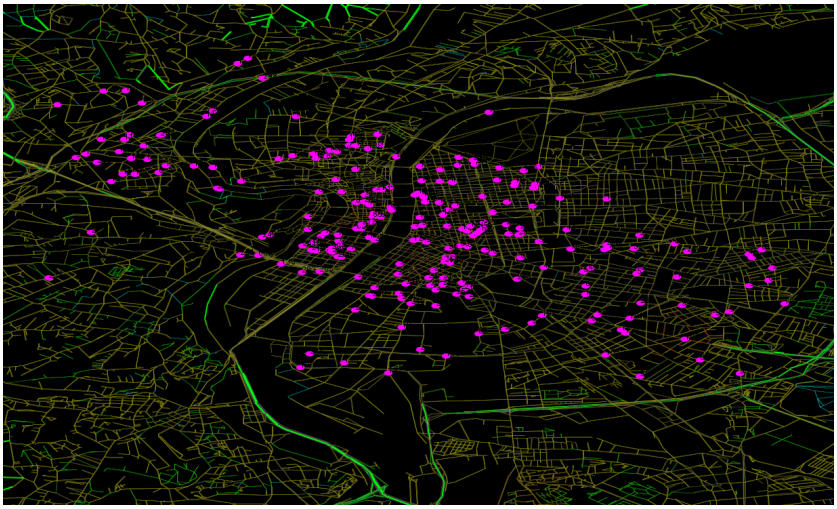

Fig. 1. 255 delivery addresses in Lyon

The travel time functions given in the benchmark are represented as stepwise functions[1]. Examples of time-dependent travel time functions generated by this approach are depicted on Fig. 2.

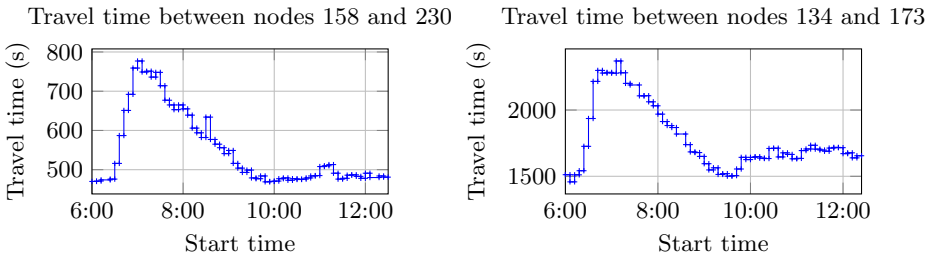

Fig. 2. Example of time-dependent travel time functions

We randomly generated 500 instances for each problem size n (10, 20 and 30 visits) by randomly selecting n locations among the 255 positions referenced in the travel time function. The duration of each visit is randomly selected in the interval [60s,300s]. Because the transition function tends to underestimate the travel time in congested areas, we generated two additional versions of the function with a dilatation of travel times of respectively 10% and 20% centered on the average travel time. So we end up with 3 functions: $T00$ (the original one) and $T10$, $T20$.

[1] This was a choice made to simplify the usage of the benchmark. We could generate piecewise linear functions from the same data by using the algorithm described in [19] in the travel time calculation section.

In a preliminary study, we want to evaluate the potential gain of using a *time-dependent* travel time model compared to a less precise TSP model that rules out time dependency. Given a stepwise travel time function T, we computed a TSP matrix $MedianTSP_T$ such that for every couple (i, j) of vertices, $MedianTSP_T(i, j)$ is the median value over the set of all time-dependent values associated with (i, j) in the stepwise travel time function T.

For each instance, we have solved the TSP defined by this median matrix $MedianTSP(T)$. Then, we have computed $cost_{MedianTSP}$, which is the cost of the TSP solution found previously when considering the time-dependent travel time function T as transition cost. We denote opt_{TDTSP} the cost of the optimal solution of the TDTSP. Obviously, $cost_{MedianTSP}$ gives an upper bound of opt_{TDTSP}. To evaluate the tightness of this bound, Fig. 3 gives the cactus plot of the relative gain defined as:

$$gain = \frac{cost_{MedianTSP} - opt_{TDTSP}}{opt_{TDTSP}}$$

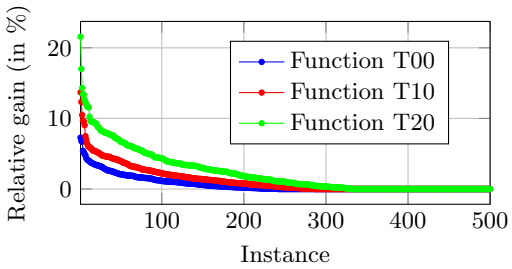

Fig. 3. Relative gain of TDTSP, instances ordered by decreasing gain

This figure shows that for more than 10% of the instances, the gain is greater than 5%, whereas for 40% of the instances it is equal to 0%. Note that a gain of 5% is considered as very important in our context. Furthermore, real-world delivery problems usually have time-window constraints. In this case, it is mandatory to consider time-dependent data. As expected, the gain tends to increase (to more than 13% and 21%) when using functions $T10$ and $T20$ with larger amplitude. A similar behavior was observed for larger problems with 20 and 30 visits although the gain was slightly smaller, probably due to the fact that the peaks of traffic congestion occur between 06:00 and 09:00 which more or less corresponds to the time frame of a 10 visit problem so, for larger problems, part of the route is executed on less congested time windows.

For each problem size n (10, 20 and 30 visits) we selected a smaller set of 60 instances that is representative of the different types of gains between TSP and TDTSP (20 instances with the largest estimated gains, 20 instances with intermediate gains, 20 instances with negligible/zero gains). These instances are used in the experimental section to compare two models for solving the TDTSP problem. The benchmark is available on liris.cnrs.fr/christine.solnon/TDTSP.html.

4 Related work

The name TSDTSP is used in the literature for two different problems. In 1978, [28] used the term TDTSP to describe the problem of scheduling jobs on a single machine with costs depending on the position of the job in the sequence. Since then, [17, 2, 8, 16] also addressed this version of the problem.

We are interested in the actual time dependent problem introduced by [26] in 1992. They give several simple heuristics for the TDTSP and for the more general Time Dependent Vehicle Routing Problem (TDVRP) where a whole fleet must be routed instead of a single vehicle. Other heuristic approaches like ant colony systems [11], monte carlo [7], tabu search [19], simulated annealing [29] and others [24, 18, 25, 13] are proposed for the TDTSP and TDVRP. Integer programming approaches are used in [8, 9, 30], some of these papers also take time-window constraints into account. To our knowledge, the only paper using a Constraint Programming approach for a time-dependent problem is [20], treating two scheduling problems with time-dependent task costs.

Aside from [14, 25], the papers cited here consider instances which are randomly generated by applying some congestion rates during rush hours. The number of time steps considered is usually 3 but can go up to 16. In most cases, not more than 4 different congestion patterns are considered. The number of visits per tour varies between 10 and 65 but optimality is rarely proven for the largest instances.

Solutions used to tackle real time-dependent vehicle routing problems in Germany are presented in [14] and in the United Kingdom in [25]. In [25] they consider 96 time steps of 15 minutes and in [14] they use 217 time steps to model a whole day but test instances are not provided. In this sense, the benchmark provided by us offers a new testing parameter for the Time-Dependent VRP/TSP. In this new benchmark, we deal with a much larger number of time steps (i.e., 65), compared to what is usually considered in existing work.

5 Classical CP model for the TDTSP and its limitations

5.1 CP model

In this section we consider that f is a step function where each (time-)step has the same length l so that f is modeled with a cost matrix T. The input data is :

- A number $n > 0$ of visits, by convention the first (resp. last) visited vertex is 1 (resp. $n + 1$).
- A time horizon $H > 0$, a number of time steps $m > 0$ and a duration $l > 0$ of time steps so that $H = lm$.
- A cost matrix $T : [1, n + 1] \times [1, n + 1] \times [0, m] \to \mathbb{R}^+$ so that the travel time from vertex i to vertex j when leaving i at time t is given by $T[i][j][t/l]$.
- A visit duration vector $D : [1, n] \to \mathbb{R}^+$.

We present here a TDTSP model adapted from the classic CP model used to solve the TSP (see [6]). We added variables $time[i]$, which give the arrival time at each vertex i, and modified constraints to take into account the fact that a duration D_i is associated with every vertex i, and that travel durations are time-dependent.

$$
\begin{aligned}
\text{intVar} \quad & position[1..n] \in 1..n \\
& next[1..n+1] \in 1..n+1, prev[1..n+1] \in 1..n+1 \\
& time[1..n+1] \in 0..H \\
\text{minimize} \quad & time[n+1] \\
\text{subject to} \quad & \text{alldifferent}(position), \text{alldifferent}(next), \text{alldifferent}(prev) \\
& \text{inverse}(prev, next) \\
& position[1] = 1, time[1] = 0, prev[1] = n+1, next[n+1] = 1 \\
& \forall i \in 1..n+1 : next[i] \neq i, prev[i] \neq i \\
& \forall i \in 1..n : position[next[i]] = position[i] + 1 \\
& \forall i \in 2..n+1 : position[prev[i]] + 1 = position[i] \\
& \forall i \in 1..n+1 : time[i] \geq time[prev[i]] + D[prev[i]] + T[prev[i]][i][time[prev[i]]/l] \quad (1) \\
& \forall i \in 1..n+1 : time[next[i]] \geq time[i] + D[i] + T[i][next[i]][time[i]/l] \quad (2)
\end{aligned}
$$

Note that the numbers of positions in a path is $n+1$ since we have to return to the depot. For each visit i: $next[i]$ and $prev[i]$ give the next and previous visits, $position[i]$ the position of the visit in the path and $time[i]$ the time of arrival at i. Beside constraints at the extremities of the tour to fix initial and end visits and the start time, an `alldifferent` constraint is posted on each group of variables whereas $prev/next$ variables are linked with `inverse` constraints. The relation between time and relative positions of visits is modeled with constraints (1) and (2). For a stronger propagation, the term T[...] in these constraints is modeled using a table constraint.

We also added the following redundant constraints to help improving the lower bound on the objective term $time[n+1]$. We noticed that these redundant constraints help reducing the number of branches by a factor close to 2 and the CPU time by a factor varying between 1 and 2.

$$time[n+1] \geq \sum_{i \in 1..n} D[i] + \sum_{i \in 1..n} T[i][next[i]][time[i]/l] \quad (3)$$

$$time[n+1] \geq \sum_{i \in 1..n} D[i] + \sum_{i \in 2..n+1} T[prev[i]][i][time[prev[i]]/l] \quad (4)$$

In the search branching scheme used to compare the performance of the propagation in section 7, we use a search that builds the sequence of visits in a chronological order. For this reason, we added a new set of variables $atPosition[j]$ that represent the vertex at the j^{th} position in the sequence. These variables are related with the rest of the model thanks to the following constraints:

$$
\begin{array}{rl}
\text{intVar} & atPosition[1..n] \in 1..n+1 \\
\text{constraints} & \text{alldifferent}(atPosition) \\
& \text{inverse}(position, atPosition) \\
& atPosition[1] = 1, next[atPosition[n]] = n+1, atPosition[n] = prev[n+1] \\
& \forall j \in 1..n : next[atPosition[j]] = atPosition[j+1] \\
& \forall j \in 1..n : prev[atPosition[j+1]] = atPosition[j]
\end{array}
$$

5.2 Model limitations

In the context of TDTSP, time variables $time[i]$ representing the dates of a visit are very important and their domain should be as tight as possible because the value of the travel time depends on the actual time value. An important limitation of the model presented above is the weakness of the propagation between temporal variables *time* and sequencing variables (like *next* and *prev*). For instance, it should be clear from their formulation that constraints like (1) and (2) would benefit from some more global reasoning over the travel time between i and $prev[i]$ (resp. between i and $next[i]$). Furthermore, reasoning only locally on direct successors of a visit (*next*, *prev*) may miss some important propagation as illustrated by the following example.

We call a visit a a *successor* of another visit b if a comes somewhere after b in the path, we call it *next* of b if it is visited exactly after b. We can see from our variables that all the propagation is done reasoning with direct neighbors of a visit (*next*, *prev*).

We can show that reasoning with successors (besides prev/next variables) allows to obtain tighter bounds on the time of visits, as soon as the problem is asymmetric[2]. For simplification we will work here with a TSP example. Consider the following slightly asymmetric TSP problem where D is the depot, A, B, C are visits and the distance matrix T is as shown on Table 1. We suppose that the upper-bound of the objective is 100, therefore only two paths are feasible (D, B, C, A, D) and (D, C, B, A, D), each with a total length of 100. Given these two feasible solutions, the tightest possible domains of *prev* and *next* variables can be seen in Table 2.

In what follows we use $dom(a)$ to refer to the domain of a variable a, $\underline{a}$ for the smallest value in its domain and $\bar{a}$ for the biggest. If a is fixed then $\underline{a} = \bar{a}$ and $dom(a)$ is a singleton.

If time bounds are computed using only *prev*/*next* variables, the best we can do boils down to apply the following formulas to compute the minimum time bounds (smallest value in the domain) until a fix point is reached:

$$
\begin{array}{l}
\underline{time[A]} = max(\underline{time[A]}, min(\underline{time[B]} + T[B][A], \underline{time[C]} + T[C][A])) \\
\underline{time[B]} = max(\underline{time[B]}, min(\underline{time[C]} + T[C][B], \underline{time[D]} + T[D][B])) \\
\underline{time[C]} = max(\underline{time[C]}, min(\underline{time[B]} + T[B][C], \underline{time[D]} + T[D][C]))
\end{array}
$$

[2] Asymmetric in the sense that reversing a solution may change its total travel time or its feasibility. Some common causes of asymmetry are: asymmetric travel times (like time-dependent travel times), time windows constraints or precedences between visits.

Table 1. Distance Matrix T

	D	A	B	C
D	0	9	46	8
A	8	0	46	8
B	46	46	0	38
C	8	8	38	0

Table 2. Next & Prev Domains

visit	dom(next)	dom(prev)
D	{B,C}	
A	{D}	{B,C}
B	{A,C}	{C,D}
C	{A,B}	{B,D}

It gives $\underline{time[A]} = 16$, $\underline{time[B]} = 46$ and $\underline{time[C]} = 8$. Let's now look at how we could propagate by also considering (indirect) successors. The distances in matrix T satisfy the triangle inequality[3] so we know that if A comes before B in the path the total length of any path starting from D, passing through A and B and returning back to D will be at least $T[D][A]+T[A][B]+T[B][D] = 101$, which is infeasible. Thus B must be visited before A and we can infer a precedence $B \rightarrow A$. This corresponds to the so-called disjunctive constraint in scheduling. With this, we know that A cannot start before $\underline{time[A]} = T[D][B] + T[B][A] = 92$, which is a lot better than the value of $\underline{time[A]} = 16$ found when reasoning only with *prev* and *next* variables.

If the TSP was purely symmetric we would not be able to deduce any successor links (indirect precedence) since any solution would be reversible and give the same cost. This type of reasoning is interesting as soon as solutions are asymmetric, which is usually the case for time-dependent travel times.

6 Time-Dependent No-Overlap constraint

In order to integrate this kind of reasoning we use the concepts of *interval* and *sequence variables* in CP Optimizer [22, 23]. Each visit i is modeled as an interval variable with start time denoted $time[i]$. The tour is modeled as a sequence variable over the set of visits. This variable maintains a precedence graph to propagate temporal relations between visits [15]. The vertices of this graph are the *time* variables associated to each visit. Two types of arcs are considered:

1. A **next arc** between two visits $i \Rightarrow j$ means that we pass through j directly after visiting i.
2. A **successor arc** between two visits $i \rightarrow j$ means that we visit j after i but we can go through other visits in between.

During the search, new **next** and **successor** arcs are added into the precedence graph because of search decisions (like when chronologically building a route) or as the result of constraint propagation (for instance by the extended disjunctive constraint sketched in subsection 6.3). The precedence graph incrementally maintains the transitive closure of the arcs.

[3] If the triangle inequality is not satisfied, one can easily pre-compute a smaller transition time corresponding to the length of the shortest path (using Floyd-Warshall algorithm) to provide a lower bound on travel times. That is what the `noOverlap` constraint of CP Optimizer is doing internally.

In CP Optimizer the `NoOverlap` constraint allows to enforce a minimal transition time between vertices on the precedence graph. In this paper, we extended the `NoOverlap` into a `TDNoOverlap` constraint to take into account time-dependent transition times. The resulting model using the new `TDNoOverlap` constraint is sketched below.

$$\begin{aligned}
\text{intervalVar} \quad & visit[i \in 1..n+1] \,\text{size}\, D[i] \\
\text{sequenceVar} \quad & tour \,\text{in all}(i \in 1..n)\ visit[i] \\
\text{minimize} \quad & \text{startOf}(visit[n+1]) \\
\text{subject to} \quad & \text{first}(tour, visit[1]) \\
& \text{last}(tour, visit[n+1]) \\
& \text{tdnooverlap}(tour, T)
\end{aligned}$$

To propagate the bounds of time variable domains we need lower bound functions for the time-dependent transition time functions. Given two visits i and j, such that there exists a next arc or a successor arc going from i to j, we define two lower bound functions for each type of arc:

1. $f^{next}_{earliest}(i,j,t_d)$ and $f^{succ}_{earliest}(i,j,t_d)$, are the transition times giving the earliest arrival time at j if we leave i at time t_d or later.
2. $f^{next}_{latest}(i,j,t_a)$ and $f^{succ}_{latest}(i,j,t_a)$, are the transition times giving the latest departure time from i in order to arrive at j at time t_a or earlier.

With these functions, the `TDNoOverlap` constraint propagates the earliest time for j (5) and the latest time for i (6), by using the adequate lower bound function depending on whether we propagate a successor arc ($x = succ$) or a next arc ($x = next$):

$$\underline{time[j]} \geq \underline{time[i]} + D[i] + f^{x}_{earliest}(i,j,\underline{time[i]} + D[i]) \tag{5}$$

$$\overline{time[i]} + D[i] \leq \overline{time[j]} - f^{x}_{latest}(i,j,\overline{time[j]}) \tag{6}$$

Now we introduce the formal definitions of the bounding functions and explain how to calculate them.

6.1 Propagation of *next* arcs

Here we consider a next arc $i \Rightarrow j$ in the precedence graph. The earliest arrival time at j, if we leave i at time t_d, is propagated by formula (5), using the transition time function:

$$f^{next}_{earliest}(i,j,t_d) = \min_{t \geq t_d}\{f(i,j,t) + t - t_d\} \tag{7}$$

In $f^{next}_{earliest}$ we check if leaving from visit i later (waiting in place) allows to arrive at j sooner. In the case where waiting is never advantageous we say that the transition times satisfy the FIFO property.

Definition 5 (FIFO property). *A time-dependent transition time function f is said to satisfy the FIFO (First In First Out) property iff:*

$$\forall i, j \in V, \forall t_1, t_2, \big(t_1 \leq t_2\big) \Rightarrow \big(t_1 + f(i,j,t_1) \leq t_2 + f(i,j,t_2)\big)$$

It follows from Def. 5 and from Eq. (7) that if f satisfies the FIFO property then $f^{next}_{earliest}$ and f are equal. Although the FIFO property generally holds in practice, our approach does not assume that f satisfies the FIFO property for two reasons: (1) stepwise functions do not satisfy it because of the discretization and (2) imprecision in data acquisition and time-dependent travel time calculations may introduce non-FIFO effects.

If f is a stepwise or a piecewise linear function, $f^{next}_{earliest}$ is a piecewise linear function. So, as we need in any case to handle piecewise linear functions in the propagation, in our implementation of the `TDNoOverlap` constraint we decided to treat the more general case where the input function f is a piecewise linear function.

In Algorithm 1 we describe the method used in pre-solve phase to calculate $f^{next}_{earliest}$ for each pair of vertices in the graph. We suppose a fixed arc (i,j) and f a piecewise linear function defined on the time domain $T = [t_{Min}, t_{Max}) \in \mathbb{R}$ and we simplify the notation to $f(t)$ instead of $f(i,j,t)$.

Each time interval $p_k = [t^k_{min}, t^k_{max})$ on which the function is linear is called a **piece**. Since p_k is open on t^k_{max}, by abuse of notation we write $f(t^k_{max})$ for $\lim_{x \to t^k_{max}} f(x)$. For instance, if f is not continuous on $t^k_{max} = t^{k+1}_{min}$ then $f(t^k_{max}) \neq f(t^{k+1}_{min})$. The notation $f \restriction_{p_k}$ means that f is restricted to interval p_k and therefore all operations are done only in this interval.

Finally, function $linear((t,v),(t',v'))$ denotes the linear function defined by the two points (t,v) and (t',v').

In the implementation, we used the class of piecewise linear function provided by CP Optimizer[4]. If ν is the number of pieces of the function, this class allows for a random access to a given piece with an average complexity of $O(log(\nu))$. Furthermore, when two consecutive pieces of the function are co-linear, these pieces are automatically merged so that the function is always represented with the minimal number of pieces.

The other type of propagation we do on next arcs in formula (6) depends on the estimation of the latest departure time from i in order to arrive at j at time t_a or before, given by:

$$f^{next}_{latest}(i,j,t_a) = \min_{t + f(i,j,t) \leq t_a} \{t_a - t\} \tag{8}$$

Since $f^{next}_{earliest}$ already gives the minimum transition time from a given time it is clear that the minimum in Equation (8) is satisfied for the biggest t' such that $t' + f^{next}_{earliest}(i,j,t') \leq t_a$. Then, calculating f^{next}_{latest} comes down to finding this biggest t'.

Algorithm 2 describes the method used in a presolve phase to compute f^{next}_{latest} for each pair of vertices in the graph.

[4] Namely: `IloNumToNumSegmentFunction`.

Algorithm 1 Calculate $f^{next}_{earliest}$

Require: f, ν (the number of pieces of f)
1: $f^{next}_{earliest} \leftarrow f$
2: **for all** $k \in \{0, ..., \nu\}$ **do**
3: $x_0 \leftarrow t^{k+1}_{min}$
4: $v_0 \leftarrow f(x_0)$
5: **if** $v_0 < f(t^k_{max})$ **then**
6: **for all** $j \in \{0, ..., \nu\} | p_j \subset [t_{Min}, x_0)$ **do**
7: $f^{next}_{earliest} \restriction_{p_j} \leftarrow \min(f^{next}_{earliest}, linear((x_0, v_0), (v_0, x_0)) \restriction_{p_j})$
8: **end for**
9: **end if**
10: **end for**
11: **return** $f^{next}_{earliest}$

Algorithm 2 Calculate f^{next}_{latest}

Require: $f^{next}_{earliest}$, ν (the number of segments of $f^{next}_{earliest}$)
1: $arrivalTime \leftarrow linear((0, 0), (1, 1)) + f^{next}_{earliest}$
2: $f^{next}_{latest}(t) \leftarrow 0$
3: **for all** $k \in \{0, ..., \nu\}$ **do**
4: $x_0 \leftarrow t^k_{min}$
5: $x_1 \leftarrow t^k_{max} - 1$
6: $v_0 \leftarrow arrivalTime(x_0)$
7: $v_1 \leftarrow arrivalTime(x_1)$
8: $slope_k(t) \leftarrow x_0 + \frac{x_1 - x_0}{v_1 - v_0} * (t - v_0)$
9: **for all** $j \in \{0, ..., \nu\} | \ p_j \subset [v_0, v_1]$ **do**
10: $f^{next}_{latest} \restriction_{p_j} \leftarrow \max(f^{next}_{latest}, linear((v_0, x_0), (v_1, x_1))) \restriction_{p_j}$
11: **end for**
12: **for all** $j \in \{0, ..., \nu\} | \ p_j \subset]v_1, arrivalTime(t_{Max})]$ **do**
13: $f^{next}_{latest} \restriction_{p_j} \leftarrow \max(f^{next}_{latest}, x_1) \restriction_{p_j}$
14: **end for**
15: **end for**
16: **return** f^{next}_{latest}

A similar procedure as the one described in Alg. 1 is used by [14], they also propose an algorithm like Alg. 2 but in their case, calculations are done for a single time t instead of calculating the whole function at once.

6.2 Propagation of *successor* arcs

Now we consider a successor arc $i \rightarrow j$ in the precedence graph. To estimate the earliest possible time of arrival at j if we leave i at time t_d or after we have to check if we can arrive faster at j by passing through other vertices. Let $\wp^{i,j}_{\tau,f}$ be the set of all timed-paths from i to j starting after time τ with travel times function f. We have:

$$f^{succ}_{earliest}(i,j,t_d) = \min_{p \in \wp^{i,j}_{t_d,f}} t(j,p) - t_d$$

where $t(j,p)$ is the start time of j in path p.

If there exists a shortest path from i to j, shorter than the direct arc, then the triangular inequality extended to the time-dependent case does not hold. It means that there is at least one vertex k such that passing through k allows to arrive faster at j.

Definition 6 (Time-dependent triangular inequality). *Function f is said to satisfy the triangular inequality property iff:*

$$\forall i,j,k \in V, \forall t \in \mathbb{R}^+, f(i,k,t) \leq f(i,j,t) + f(j,k,t + f(i,j,t))$$

To calculate $f^{succ}_{earliest}$ we use a time-dependent extension of the Floyd Warshall All Pairs Shortest Path algorithm [10]. We use $f^{next}_{earliest}$ as travel time function in the algorithm so that waiting at intermediate vertices to possibly go faster is already taken into account.

The second type of propagation on successor arcs is based on the estimation of the latest departure time from i in order to arrive at j at time t_a or before, given by:

$$f^{succ}_{latest}(i,j,t_a) = \min_{p \in \wp^{i,j}_{t,f}, t(j,p) \leq t_a} t_a - t$$

The reasoning for calculating f^{succ}_{latest} is exactly the same as the one we used for f^{next}_{latest} and the algorithm (2) is the same too, only this time we use $f^{succ}_{earliest}$ instead of $f^{next}_{earliest}$ as input.

6.3 Time-dependent disjunctive propagation

Classical propagation algorithms used in constrained-based scheduling can be extended to time-dependent transition times. In our implementation of the `TDNoOverlap` constraint we extended the disjunctive reasoning [5]. As soon as two visits i and j are such that one of the conditions below is satisfied then it is clear that it is not possible to visit j before i and thus, we can add a successor arc $i \rightarrow j$ in the precedence graph:

$$\underline{time[j]} + D[j] + f^{succ}_{earliest}(j,i,\underline{time[j]} + D[j]) > \overline{time[i]}$$

$$\overline{time[i]} - f^{succ}_{latest}(j,i,\overline{time[i]}) - D[j] < \underline{time[j]}$$

This extended disjunctive reasoning helps discovering new arcs in the precedence graph that are themselves propagated as described in subsections 6.1 and 6.2.

6.4 Complexity

The complexity of the `TDNoOverlap` constraint is dominated by the complexity of maintaining the precedence graph and the disjunctive propagation. The worst-case complexity of the full-fledged propagation is quadratic with respect to the number of visits. We also implemented a slightly weaker but lighter propagation with linear complexity.

7 Experimental evaluation

We compare the classical CP model presented in section 5 with the model using the `TDNoOverlap` constraint[5] presented in section 6.

In a first experiment we compare the filtering power of the two models. We use the same depth first search strategy for both models so that we can estimate the impact of constraint propagation on the number of branches of the complete search tree. We do a chronological scheduling of visits and choice of the nearest visit in terms of transition time first, given that the earliest date of the previous visit is known. For the classical CP model, this means that the search first fixes the variables $atPosition[i]$ for $i = 1, 2, ...n$. However, as the search strategy is not static, the search tree is different (one tree is not a sub-tree of the other). We could have tested on a static search strategy but this would have resulted in a more "artificial" type of search. We measured the number of branches and the CPU time of the two approaches on the 60 instances of size 10 on the 3 functions $T00$, $T10$, $T20$[6]. For those 180 tests (all solved to optimality with both models), the left side of Fig. 4 shows the comparison of the number of branches of the search tree explored by the two approaches while the right side compares the CPU times.

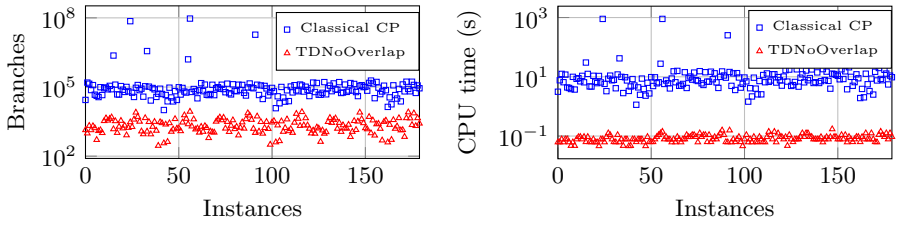

Fig. 4. Comparison of number of branches (left) and CPU time (right)

Not only the `TDNoOverlap` model propagates a lot more (about 50 times fewer branches) but it also finds better solutions faster than the classical CP model. For 10-sized instances the search is about 100 times faster on average.

On Fig. 5 we compare on the instances of size 20 and 30 the cost of the best solution found by the two approaches using the automatic search of CP

[5] In these experiments we used the lighter version of the propagation but we noticed that there was not much difference with respect to the full-fledged version.

[6] Comparison was performed only on instances of size 10 as the classical CP model is not able to solve the larger problems to optimality.

Optimizer which is more sophisticated than depth first search. We used the same search heuristics as above and a time limit of 900s.

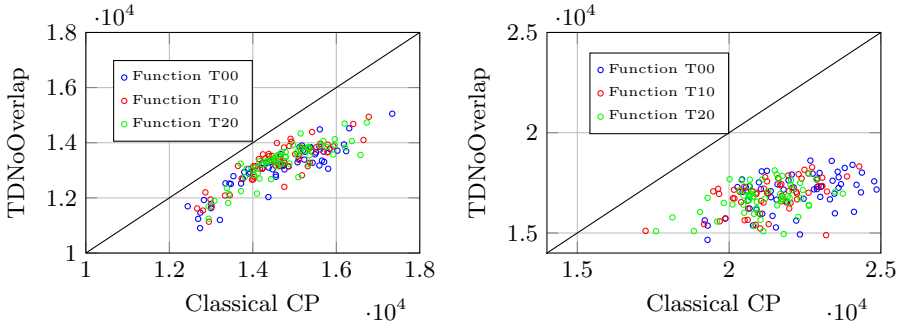

Fig. 5. Comparison of solution quality on problems of size 20 (left) and 30 (right)

For instances of size 20, the `TDNoOverlap` model finds and proves the optimal solution for 165 instances out of 180 and, in average the solution found is more than 10% better than the one of the classical CP model. For instances of size 30, both models are incapable of proving optimality in the time limit, but in average the solutions found by `TDNoOverlap` are more than 20% better than the ones of the classical CP model.

We also compared the `TDNoOverlap` model with an intermediate model (described in [3]) using the `NoOverlap` constraint. In average on the 180 intances of size 10, the `TDNoOverlap` model proves optimality with 20 times less branches and is 40 times faster.

8 Discussion

In this paper we showed, in the context of scheduling a sequence of deliveries, the impact of reasoning with successors, other than just next/previous, and of taking time-dependent transition times into account directly into a global constraint. Reasoning on successors is crucial for problems involving time variables like the TDTSP. From an application perspective, the interest of the scheduling model we presented is that it is very easy to integrate additional constraints like precedence between visits or disjunctive time-windows. These constraints are in fact directly available in CP Optimizer and should work pretty well when added to the central TDTSP model presented in this paper.

Reasoning on successors is in fact complementary with reasoning on a prev/next graph. In future work we plan to improve our constraint propagation by calculating tighter bounds for the TDTSP by using Minimum Spanning Trees or Assignment Problem relaxations on the prev/next graph, extending the approaches described in [15, 6, 12]. We also want to see if the successor relations stored in the precedence graph can be exploited in this context. We plan to evaluate our new constraint on other benchmarks, and compare it with other approaches.

Acknowledgments

This work has been done in the context of the Optimod'Lyon project. We would like to give our special thanks to Thomas Baudel for his help in the obtention of traffic data.

Christine Solnon is supported by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

References

[1] Optimod'Lyon website (21/07/2014), http://www.optimodlyon.com/
[2] Abeledo, H., Fukasawa, R., Pessoa, A., Uchoa, E.: The time dependent traveling salesman problem: polyhedra and algorithm. Mathematical Programming Computation 5(1), 27–55 (2013)
[3] Aguiar Melgarejo, P., Baudel, T., Solnon, C.: Global and reactive routing in urban context: first experiments and difficulty assessment. In: CP-2012 Workshop on Optimization and Smart Cities (Oct 2012)
[4] Baptiste, P., Laborie, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling and planning. In: F. Rossi, P. van Beek, T.W. (ed.) Handbook of Constraint Programming, chap. 22, pp. 759–798. Elsevier (2006)
[5] Baptiste, P., Le Pape, C.: Disjunctive constraints for manufacturing scheduling: Principles and extensions. In: Proc. 3rd International Conference on Computer Integrated Manufacturing (1995)
[6] Benchimol, P., van Hoeve, W.J., Regin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints pp. 205–233 (2012)
[7] Bentner, J., Bauer, G., Obermair, G.M., Morgenstern, I., Schneider, J.: Optimization of the time-dependent traveling salesman problem with Monte Carlo methods. Physical Review E 64(3) (Aug 2001)
[8] Bront, J.: Integer Programming approaches to the Time Dependent Travelling Salesman Problem. Ph.D. thesis, Universidad de Buenos Aires (2012)
[9] Cordeau, J.F., Ghiani, G., Guerriero, E.: Properties and Branch-and-Cut Algorithm for the Time-Dependent Traveling Salesman Problem
[10] Cormen, T.H., Leiserson, C.E., L., R.R.: Introduction to Algorithms. MIT Press, McGraw-Hill edn. (1990)
[11] Donati, A.V., Montemanni, R., Casagrande, N., Rizzoli, A.E., Gambardella, L.M.: Time dependent vehicle routing problem with a multi ant colony system. European Journal of Operational Research 185(3), 1174–1191 (Mar 2008)
[12] Fages, J.G., Lorca, X.: Improving the asymmetric tsp by considering graph structure. arxiv preprint arxiv:1206.3437 (2012)
[13] Figliozzi, M.A.: The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. Transportation Research Part E Logistics and Transportation Review 48, 616–636 (2012)
[14] Fleischmann, B., Gietz, M., Gnutzmann, S.: Time-Varying Travel Times in Vehicle Routing. Transportation Science 38(2), 160–173 (May 2004)
[15] Focacci, F., Laborie, P., Nuijten, W.: Solving Scheduling Problems with Setup Times and Alternative Resources. AIPS Proceedings (2000)

[16] Fox, K.R., Gavish, B., Graves, S.C.: An n-Constraint Formulation of the (Time-Dependent) Traveling Salesman Problem. Operations Research 28(4), 1018–1021 (1980)
[17] Gouveia, L., Voss, S.: A classification of formulations for the (time-dependent) traveling salesman problem. European Journal of Operational Research 83(1), 69 – 82 (1995)
[18] Hashimoto, H., Yagiura, M., Ibaraki, T.: An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. Discrete Optimization 5(2), 434–456 (May 2008)
[19] Ichoua, S., Gendreau, M., Potvin, J.Y.: Vehicle Dispatching With Time-Dependent Travel Times. European Journal of Operations Research (2003)
[20] Kelareva, E., Tierney, K., Kilby, P.: CP Methods for Scheduling and Routing with Time-Dependent Task Costs. EURO Journal on Computational Optimization 2, 147–194 (2014)
[21] Laborie, P., Godard, D.: Self-adapting large neighborhood search: Application to single-mode scheduling problems. In: Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA). pp. 276–284 (2007)
[22] Laborie, P., Rogerie, J.: Reasoning with Conditional Time-Intervals. In: FLAIRS Conference. pp. 555–560 (2008)
[23] Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Reasoning with Conditional Time-Intervals. Part II: an Algebraical Model for Resources. In: FLAIRS Conference (2009)
[24] Li, F., Golden, B., Wasil, E.: Solving the time dependent traveling salesman problem. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Next Wave in Computing, Optimization, and Decision Technologies, Operations Research/Computer Science Interfaces Series, vol. 29, pp. 163–182. Springer US (2005)
[25] Maden, W., Eglese, R., Black, D.: Vehicle Routing and Scheduling with Time Varying Data: A Case Study. Journal of the Operational Research Society 61, 515–522 (2009)
[26] Malandraki, C., Daskin, M.S.: Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. Transportation Science 26, 185–200 (1992)
[27] Nannicini, G.: Point-to-Point Shortest Paths on Dynamic Time-Dependent Road Networks. Ph.D. thesis, Ecole Polytechnique, Palaiseau (2009)
[28] Picard, J.C., Queyranne, M.: The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling (1978)
[29] Schneider, J.: The time-dependent traveling salesman problem. Physica A: Statistical Mechanics and its Applications 314(1-4), 151–155 (Nov 2002)
[30] Stecco, G., Cordeau, J.F., Moretti, E.: A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. Computers & Operations Research 35(8), 2635 – 2655 (Aug 2007)