Computers & Operations Research 54 (2015) 168–176

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Solving the Orienteering Problem with Time Windows via the Pulse Framework

Daniel Duque, Leonardo Lozano, Andrés L. Medaglia *

Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Cr 1E No. 19A-10, ML711 Bogotá, Colombia

ARTICLE INFO

Available online 28 September 2014

Keywords:
Routing
Shortest path problems with side constraints
Traveling salesman problem with profits
Vehicle routing problem with time windows

ABSTRACT

The Orienteering Problem with Time Windows (OPTW) is the problem of finding a path that maximizes the profit available at the nodes in a time-constrained network. The OPTW has multiple applications in transportation, telecommunications, and scheduling. First, we extend an exact method for shortest path problems with side constraints into a general-purpose framework for hard shortest path variants. Then, using this framework, we develop a new method for the OPTW that incorporates problem-specific knowledge. Our method outperforms the state-of-the-art algorithm on instances derived from benchmark datasets from the literature achieving speedups of up to 266 times and is able to find optimal solutions to large-scale problems with up to 562 nodes in short computational times.

1. Introduction

The *Orienteering Problem with Time Windows* (OPTW) is defined over a directed graph $\mathcal{G}=(\mathcal{N},\mathcal{A})$, where $\mathcal{N}=\{v_1,\ldots,v_n\}$ is the set of nodes, $\mathcal{A}=\{(i,j)|v_i\in\mathcal{N},v_j\in\mathcal{N},i\neq j\}$ is the set of arcs, and v_s and v_e denote the start and end nodes, respectively. Each node $v_i\in\mathcal{N}$ has a score s_i, a service time e_i, and a time window when the service can take place $[a_i,b_i]$. However, it is possible to visit a node before a_i and wait to begin the service at time a_i. For all arcs $(i,j)\in\mathcal{A}$, let there be a nonnegative weight $t_{i,j}$ that refers to the travel time from node $v_i\in\mathcal{N}$ to node $v_j\in\mathcal{N}$ that includes the service time e_i of node v_i and satisfies the triangle inequality. The OPTW is the problem of finding a path $\mathcal{P}$ (i.e., an ordered set of nodes) from v_s to v_e that collects the maximum amount of score subject to a maximum travel time T and satisfies the time window at every node $v_i\in\mathcal{P}$. Henceforth, we use functions $s(\cdot)$ and $t(\cdot)$ to represent the score and the time of any given path $\mathcal{P}$. We denote by $\mathcal{P}_{i,j}=\{v_i,\ldots,v_{(k)},\ldots,v_j\}$ a path that starts at node $v_i\in\mathcal{N}$ and ends at node $v_j\in\mathcal{N}$ where $v_{(k)}$ represents the node at the k-th position of the sequence.

The OPTW naturally arises in a wide range of transportation and logistics applications. While planning a trip, a tourist might want to visit a list of places (e.g., cultural venues), but it is not possible to visit them all due to time limitations [28]. Similar to the tourist trip design, there are military applications that require planning tactical aerial or naval reconnaissance operations. In this context, an artifact is routed through several locations with the goal of maximizing the overall surveillance [21,32,36,25]. Similar concepts have been applied in the field of vehicle routing, where a set of tasks performed by technicians need to be planned within a daily schedule [29,30]. Tricoire et al. [31] considered the problem where sales people have to visit a set of customers (within a time interval), but visiting them all is not possible within a fixed time horizon (e.g., a workday). In a production environment, Ílhan et al. [17] presented a supplier visiting problem where a manufacturer decides which suppliers to audit in order to maximize monetary claims associated with inventory leftovers.

Both exact and heuristic solution strategies have been developed for the *Orienteering Problem* (OP). Among the exact approaches, Fischetti et al. [12] and Gendreau et al. [14] used valid inequalities in a *branch-and-cut* procedure to solve instances with up to 500 nodes. In contrast to the exact methods, there are plenty of heuristics available in the literature. Golden et al. [16] proposed a heuristic based on a sampling method. Ramesh and Brown [22], and Chao et al. [6] used insertion heuristics (e.g., cheapest insertion) aided by 2-Opt and 3-Opt procedures. Gendreau et al. [15] proposed a tabu search that inserts and removes sets of nodes iteratively. More recently, Schilde et al. [26] developed a metaheuristic for the biobjective OP that outperforms several heuristics for the single-objective OP.

Despite the vast number of approaches available for the OP, few researchers have dealt with the OPTW. Righini and Salani [24] extended a bidirectional dynamic programming (DP) algorithm initially developed for the *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) [23]. They used a technique named Decremental State Space Relaxation (DSSR) in which the DP algorithm takes advantage of a state space reduction. This state space reduction arises as a relaxation of the elementarity condition over a subset of nodes,

* Corresponding author. Tel.: +57 1 3394949x2880.
E-mail address: amedagli@uniandes.edu.co (A.L. Medaglia).
URL: http://wwwprof.uniandes.edu.co/~amedagli (A.L. Medaglia).

http://dx.doi.org/10.1016/j.cor.2014.08.019
0305-0548/

which in turn reduces the dimension of the state vector and the state space. For the OPTW, they applied the state space relaxation over the state variables related to the node visits and, in exchange, the total number of visited nodes is considered as a state variable. Montemanni and Gambardella [20] solved the OPTW using an ant colony (AC) heuristic and improved the best known solutions for some instances from the literature. Additionally, they tackled the *Team Orienteering Problem with Time Windows* (TOPTW), which seeks a set of routes instead of a single one. Gambardella et al. [13] presented an enhanced AC heuristic that uses a local search procedure. A similar heuristic based on AC with local search was proposed by Verbeeck et al. [35]. Vansteenwegen et al. [34] also solved the TOPTW, but using an iterated local search approach, while Tricoire et al. [31] solved a multi-period OPTW with a variable neighborhood search heuristic. More recently, Archetti et al. [1] proposed a branch-and-price procedure to tackle a close variant of the TOPTW known as the *Capacitated Team Orienteering Problem* (*CTOP*) which instead of the time windows considers a constraint related to the vehicles capacity. The subproblem emerging from the column generation procedure is solved using a DP algorithm with the acceleration strategies proposed by Righini and Salani [24] and the 2-cycle elimination procedure presented by Desrochers et al. [8]. This branch-and-price method has been extended for other variants of routing problems with profits by Archetti et al. [2,3]. For a detailed review of the OP and its variants (including the OPTW), we refer the reader to the survey presented by Vansteenwegen et al. [33].

In this paper, we extend the pulse algorithm for shortest paths with side constraints [19,18] and present it as a general-purpose *pulse framework* for hard shortest path problems. This framework is based on the idea of performing an implicit enumeration of the entire solution space supported by *pruning strategies* that efficiently discard a vast number of suboptimal and infeasible solutions. The framework relies on *core* components that can be easily extended for different hard shortest path variants and *problem-specific* (modular) components that are based on particular characteristics of the problem at hand.

The main contributions of this work are the following:

- introduces the pulse framework as a general-purpose framework for hard shortest path problems that appear as subproblems of hard routing problems;
- extends the pulse framework to build a top-performer algorithm for the OPTW;
- presents a *soft dominance* pruning strategy that exploits dominance relations between solutions in a novel label-free fashion;
- develops a *detour* pruning strategy that incorporates problem specific conditions (namely time windows) and extends it into a preprocessing technique that reduces the size of the network (i.e., arc deletion);
- presents a parallel version of the algorithm that takes advantage of multi-core architectures; and
- assesses the strength of the new pruning strategies, the effects of the parallelization, and the performance of the proposed algorithm against the state-of-the-art algorithm by Righini and Salani [24].

The remainder of this paper is organized as follows. Section 2 presents an overview of the pulse framework. Section 3 shows the application of the pulse framework and the pruning strategies for the OPTW. Section 4 describes the parallelization process. Section 5 presents the computational experiments. Finally, Section 6 concludes the paper and outlines future work.

2. An overview of the pulse framework

Given a network, the pulse algorithm sends a *pulse* from the start node v_s to the end node v_e. This pulse recursively traverses the network carrying the information of the explored partial path $\mathcal{P}$, i.e., attributes of the path such as its cumulative resource consumption and its cumulative objective function. Every time a pulse reaches the end node v_e a feasible solution is obtained and the pulse recursively backtracks to continue its propagation through the rest of the nodes. If the pulse is let free, this recursive propagation leads to a complete enumeration of all possible paths from v_s to v_e, which guarantees that the optimal solution is always found. However, the algorithm avoids complete enumeration by stopping the exploration of any partial path whenever there is enough evidence that shows that the partial path will not lead to a feasible or an improved solution (i.e., a new primal bound). To do so, several pruning strategies determine as early as possible when a partial path should be discarded. This look-ahead mechanism prunes aggressively vast regions of the solution space, turning the complete enumeration into an implicit enumeration.

The pulse algorithm can be seen as a general framework that is extended to solve a wide range of shortest path problems that are often used as components to solve hard routing problems. Different pruning strategies can be devised to exploit specific knowledge of the problem at hand. Fig. 1 shows the components of the pulse framework and how they have been implemented or extended for different shortest path problems.

The Constrained Shortest Path Problem (CSP) was the first problem tackled with the pulse algorithm [19]. For the CSP, Lozano and Medaglia [19] proposed three pruning strategies: *dominance*, *infeasibility*, and *bounds*. In Lozano et al. [18], infeasibility and bounds pruning strategies were extended to the ESPPRC, with the addition of a dominance-based problem-specific strategy called *rollback*. The key element in this work was a bounding scheme that shares the same spirit of a state space reduction in dynamic programming. By efficiently solving the ESPPRC, the pulse algorithm was successfully used as an engine to solve auxiliary problems under a column generation scheme for the Vehicle Routing Problem with Time Windows (VRPTW). Duque et al. [10] handled the Biobjective Shortest Path Problem (BSP), implementing the dominance pruning strategy from the CSP and proposing two new problem-specific pruning strategies: *nadir point* and *efficient set*. More recently, Bolívar et al. [5] proposed three acceleration strategies for the pulse algorithm applied to the Weight Constrained Shortest Path Problem with Replenishment (WCSPP-R). For the OPTW we implemented the core pruning strategies developed for the ESPPRC and added two new problem-specific pruning strategies, namely *soft dominance* and *detour*. The next section describes in detail how we adapted the framework to tackle the OPTW.

3. Pulse framework for the OPTW

The intuition behind the pulse algorithm explained in Section 2 remains unchanged for the OPTW. For this problem, a pulse traveling through the network carries out the information of the cumulative time and the cumulative score of a partial path. Algorithm 1 presents an overview of the pulse algorithm for the OPTW. Lines 1–3 initialize $\mathcal{P}$, $s(\mathcal{P})$, and $t(\mathcal{P})$. Line 4 calculates dual bounds on the maximum score that can be achieved by any partial solution. Line 5 invokes the recursive function `pulse` starting the propagation from node v_s. Line 6 returns the optimal path found throughout the recursion.

Algorithm 1. Pulse algorithm for the OPTW.

Input: $\mathcal{G}$, directed graph; v_s, start node; v_e, end node.
Output: $\mathcal{P}^*$, optimal path.

```
1: P ← {}
2: s(P) ← 0
3: t(P) ← 0
4: boundCalculation (G)          ▷see Section 3.1.2
5: pulse (v_s, s(P), t(P), P)    ▷see Algorithm 2
6: return P*
```

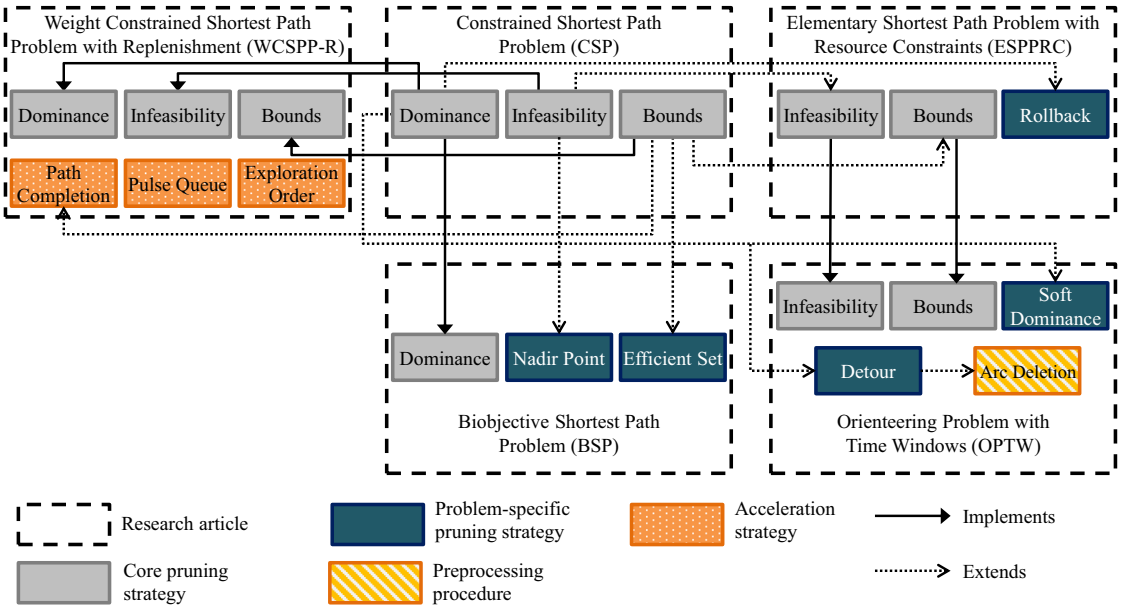

Fig. 1. Pruning strategies of the pulse framework applied to different routing problems.

Algorithm 2 shows the recursive function pulse for the OPTW. Lines 1–3 use the pruning strategies to evaluate the incoming pulse to node v_i. If the pulse is not pruned, lines 4 and 5 add the current node to the partial path and update the cumulative score $s(\mathcal{P}')$. Lines 6–11 propagate the pulse by invoking the function pulse over all nodes $v_j \in \Gamma^+(v_i)$, where $\Gamma^+(v_i) = \{v_j \in \mathcal{N} | (i,j) \in \mathcal{A}\}$ is the set of head nodes of the outgoing arcs from node v_i. Additionally, line 7 uses a strategy that prunes outgoing pulses, provided it is possible to detour before reaching node v_j. Every time the function `pulse` is invoked on the final node v_e, the information of the best path (incumbent) is updated and the pulse is no longer propagated.

Algorithm 2. Pulse function for the OPTW.

Input: v_i, current node; $s(\mathcal{P})$, cumulative score; $t(\mathcal{P})$, cumulative time; $\mathcal{P}$, partial path.
Output: `void`

```
1:  if ¬checkInfeasibility(v_i, t(P)) then                ▷see Section 3.1.1
2:    if ¬checkBounds(v_i, s(P), t(P)) then               ▷see Section 3.1.2
3:      if ¬checkSoftDominance(v_i, t(P), P) then         ▷see Section 3.2.1
4:        P' ← P ∪ {v_i}
5:        s(P') ← s(P) + s_i
6:        for v_j ∈ Γ+(v_i) do
7:          if ¬checkDetour(v_i, v_j, P, t(P)) then     ▷see Section 3.2.2
8:            t(P') ← t(P) + t_ij
9:            pulse(v_j, s(P'), t(P'), P')
10:         end if
11:       end for
12:     end if
13:   end if
14: end if
15: return void
```

The performance of the pulse algorithm depends on the capacity to prune partial paths as early as possible. The following subsections describe in detail the pruning strategies used for the OPTW.

3.1. Core pruning strategies

Core pruning strategies are independent of problem-specific knowledge and can be used in a wide range of routing problems. These strategies can be seen as the core components of the pulse framework.

3.1.1. Pruning by infeasibility

Every time the `pulse` function is invoked at node $v_i \in \mathcal{N}$, the algorithm checks if visiting v_i creates a cycle, exceeds the upper time window b_i, or violates the global time constraint T. If any of these events happens, the pulse can be safely pruned because the partial path $\mathcal{P}_{s,i}$ is already infeasible. Cycles are pruned using a binary array that stores the information of visited nodes. If $t(\mathcal{P}_{s,i}) > b_i$, then the pulse is pruned because the cumulative time of the path falls outside the time window for node $v_i \in \mathcal{N}$. For the global time constraint, if $t(\mathcal{P}_{s,i}) + t_{i,e} > T$, then the partial path is pruned because it is not possible to reach the end node v_e on time.

Finally, if v_i is reached before a_i, $t(\mathcal{P}_{s,i})$ is set to a_i, meaning that it is possible to wait until the node is able to serve the arriving entity as early as the time window opens.

3.1.2. Pruning by bounds

We use a primal bound $\underline{s}$ to prune partial solutions that promise no improvement. Every time a feasible solution is found, this primal bound is updated to keep the best objective value at hand. A dual

bound for a partial path $\mathcal{P}_{s,i}$ from v_s to v_i, $\overline{s}(\mathcal{P}_{s,i})$, refers to the maximum extra score that can be collected by completing the path in the best possible way. With this bound, if $s(\mathcal{P}_{s,i})+\overline{s}(\mathcal{P}_{s,i}) \leq \underline{s}$, the pulse can be pruned because completing partial path $\mathcal{P}_{s,i}$ would not improve the best solution found so far. However, note that finding $\overline{s}(\mathcal{P}_{s,i})$ requires solving the OPTW starting at any node v_i for every possible partial path $\mathcal{P}_{s,i}$. To avoid this difficulty, we extend the bounding scheme presented in Lozano et al. [18], where the dual bounds are conditioned to the node which the partial path $\mathcal{P}_{s,i}$ is reaching (v_i) and its cumulative time $t(\mathcal{P}_{s,i})$. These conditional dual bounds $\overline{s}(v_i, t(\mathcal{P}_{s,i}))$ denote the maximum score that can be achieved by a partial path $\mathcal{P}_{s,i}$ that is reaching node v_i and has consumed $t(\mathcal{P}_{s,i})$ units of time. Furthermore, time consumption is discretized in steps of size Δ. For every discrete level of time, the `pulse` function is invoked for all $v_i \in \mathcal{N}$ to calculate $\overline{s}(v_i, t(\mathcal{P}_{s,i}))$. This conditional bounding proceeds as follows. First, we start by calculating a bound for every node v_i with a time consumption $t(\mathcal{P}_{s,i}) \leftarrow T-\Delta$, i.e., an overly constrained OPTW starting at v_i that only has Δ time units left. In a backward mode, we calculate bounds for every node with less time consumption up to a threshold $\underline{t}$, i.e., $t(\mathcal{P}_{s,i}) \leftarrow T-2\Delta, \ldots, t(\mathcal{P}_{s,i}) \leftarrow T-\beta\Delta, \ldots, t(\mathcal{P}_{s,i}) \leftarrow \underline{t}$, where β is the number of steps of size Δ in the bounding calculation. The optimal solution found at every step is a dual bound on the maximum score for any partial path that reaches node v_i with a time consumption $t(\mathcal{P}_{s,i})$ within $[T-\beta\Delta, T-(\beta-1)\Delta)$ for any discrete step $\beta = 1, \ldots, \lceil (T-\underline{t})/\Delta \rceil$. When the bounding procedure finishes, the `pulse` function is triggered at v_s with $t(\mathcal{P}) \leftarrow 0$. In this final pulse invocation, all the bounds calculated from $\underline{t}$ to T are ready to be used for pruning. For detailed information about this bounding scheme, the reader is referred to Lozano et al. [18].

3.2. Problem-specific pruning strategies

The core pruning strategies provide a starting point to solve the OPTW, however, to improve the pulse algorithm performance it is critical to exploit particular properties of the problem at hand. For the OPTW, we propose two new pruning strategies related to the time windows and the independency between the score collection and the sequence of nodes visited in a path of the OPTW.

3.2.1. Pruning by soft dominance

This pruning strategy receives its name from the dominance tests used by DP algorithms in the context of vehicle routing [11,23]. These dominance tests save paths to a given node v_i using labels that store the objective function and the usage of resources (time consumption and visited nodes); and check for dominance relations within labels stored at any given node v_i (cf. [11]). Let $\mathcal{N}(\cdot)$ be a function that represents the (unordered) set of nodes for path $\mathcal{P}_{s,i}$. In a DP setting, a label associated with path $\mathcal{P}^1_{s,i}$ dominates the label of path $\mathcal{P}^2_{s,i}$, if the former path: (1) has the same or better score (i.e., $s(\mathcal{P}^1_{s,i}) \geq s(\mathcal{P}^2_{s,i})$); (2) consumes the same or less resource (i.e., $t(\mathcal{P}^1_{s,i}) \leq t(\mathcal{P}^2_{s,i})$); (3) visits the same nodes or a subset (i.e., $\mathcal{N}(\mathcal{P}^1_{s,i}) \subseteq \mathcal{N}(\mathcal{P}^2_{s,i})$; and (4) at least one of the three previous conditions holds strictly (i.e., $s(\mathcal{P}^1_{s,i}) > s(\mathcal{P}^2_{s,i}), t(\mathcal{P}^1_{s,i}) < t(\mathcal{P}^2_{s,i})$ or $\mathcal{N}(\mathcal{P}^1_{s,i}) \subset \mathcal{N}(\mathcal{P}^2_{s,i})$). The main drawback for using dominance tests is the space required to store a huge amount of labels holding the information of all nondominated paths that have reached any given node. Since the number of states grows exponentially with the size of the network, managing the set of labels can result computationally unbearable leading to the well-known curse of dimensionality in DP.

As an alternative to traditional dominance tests, we propose a label-free strategy that prunes a partial path $\mathcal{P}_{s,i}$ if it finds an alternative partial path $\mathcal{P}'_{s,i}$ with $t(\mathcal{P}'_{s,i}) < t(\mathcal{P}_{s,i})$ and $\mathcal{N}(\mathcal{P}'_{s,i}) = \mathcal{N}(\mathcal{P}_{s,i})$, hence, $s(\mathcal{P}'_{s,i}) = s(\mathcal{P}_{s,i})$. To find such $\mathcal{P}'_{s,i}$, we swap nodes in $\mathcal{P}_{s,i}$ between the last added node (the node visited before v_i) and the other nodes in the sequence except for v_s. If any of these swaps in $\mathcal{P}_{s,i}$ turns out to be a feasible partial path $\mathcal{P}'_{s,i}$ that consumes less time than $\mathcal{P}_{s,i}$ (i.e., $t(\mathcal{P}'_{s,i}) < t(\mathcal{P}_{s,i})$), then the path $\mathcal{P}'_{s,i}$ dominates path $\mathcal{P}_{s,i}$ and the pulse associated with $\mathcal{P}_{s,i}$ can be safely pruned. Fig. 2 shows an example of all the swaps evaluated over a partial path from v_s to v_i that has visited five intermediate nodes.

Notice that by exploring this (swap) neighborhood some dominated paths (pulses) may go through node v_i, but most importantly, the optimal solution is never pruned and yet some dominated paths are discarded. The pulse framework leaves open the possibility to implement more complex neighborhood search procedures for $\mathcal{P}'_{s,i}$, but a simple one often offers a good compromise between the strength of the pruning strategy and the computational effort. Since we do not focus in finding the best routing for the nodes in $\mathcal{N}(\mathcal{P}_{s,i})$, finding $\mathcal{P}'_{s,i}$ becomes a soft dominance test over $\mathcal{P}_{s,i}$. Nevertheless, the novelty of this strategy relies on the fact that we test dominance without using labels or any other kind of storage for all possible routing sequences of $\mathcal{N}(\mathcal{P}_{s,i})$. Finally, note that this pruning strategy exploits a very specific OPTW condition where the collected score (objective function) of a set $\mathcal{N}(\mathcal{P}_{s,i})$ is independent of its routing.

3.2.2. Pruning by detour

Consider a partial path $\mathcal{P}_{s,i}$ that is currently at node $v_i \in \mathcal{N}$ and is propagating through arc $(v_i, v_j) \in \mathcal{A}$. If the time to reach $v_j \in \mathcal{N}$ is $t(\mathcal{P}_{s,i})+t_{i,j} \leq a_j$, the algorithm checks if there is a node $v_k \in \mathcal{N}$ such that the detour $\mathcal{P}_{s,i} \cup \{v_k\} \cup \{v_j\}$ is feasible and the arrival time to node v_j is still less than a_j. If visiting v_k is feasible (i.e., $t(\mathcal{P}_{s,i})+t_{i,k} \leq b_k$) and v_j is still reached before a_j (i.e., $t(\mathcal{P}_{s,i})+t_{i,k}+t_{k,j} \leq a_j$), we can safely prune the incoming pulse to v_j since there is at least another path to v_j, $\mathcal{P}'_{s,j} = \mathcal{P}_{s,i} \cup \{v_k\} \cup \{v_j\}$, with a better score (i.e., $s(\mathcal{P}'_{s,j}) > s(\mathcal{P}_{s,i} \cup \{v_j\})$) and the same resource consumption up to node v_j (i.e., $t(\mathcal{P}'_{s,j}) \leq a_j$). To apply this strategy, we calculate for every arc (v_i, v_j) and for every detour to any node v_k the latest time that a path can arrive to v_i in order to detour to v_k and reach v_j before a_j; we denote these latest times by $L^k_{(i,j)}$. We also define $L^k_{(k,j)} \triangleq \min\{a_j - t_{k,j}, b_k\}$, the latest time a partial path can arrive to v_k in order to reach v_j before a_j. If $L^k_{(k,j)} \geq a_k$, then the latest time a partial path can arrive to v_i is $L^k_{(i,j)} \triangleq \min\{L^k_{(k,j)} - t_{i,k}, b_i\}$. Fig. 3 presents a graphical representation of the latest

$$\mathcal{P}_{s,i} \leftarrow \{v_s\} \cup \{v_{(1)}, v_{(2)}, v_{(3)}, v_{(4)}, v_{(5)}\} \cup \{v_i\}$$

$$\mathcal{P}'_{s,i} \leftarrow \{v_s\} \cup \{\boldsymbol{v_{(5)}}, v_{(2)}, v_{(3)}, v_{(4)}, \boldsymbol{v_{(1)}}\} \cup \{v_i\}$$

$$\mathcal{P}'_{s,i} \leftarrow \{v_s\} \cup \{v_{(1)}, \boldsymbol{v_{(5)}}, v_{(3)}, v_{(4)}, \boldsymbol{v_{(2)}}\} \cup \{v_i\}$$

$$\mathcal{P}'_{s,i} \leftarrow \{v_s\} \cup \{v_{(1)}, v_{(2)}, \boldsymbol{v_{(5)}}, v_{(4)}, \boldsymbol{v_{(3)}}\} \cup \{v_i\}$$

$$\mathcal{P}'_{s,i} \leftarrow \{v_s\} \cup \{v_{(1)}, v_{(2)}, v_{(3)}, \boldsymbol{v_{(5)}}, \boldsymbol{v_{(4)}}\} \cup \{v_i\}$$

Fig. 2. Example of all possible swaps under the soft dominance test for a 6-node partial path reaching node v_i.

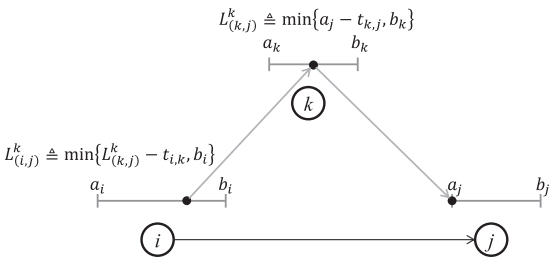

Fig. 3. Graphical representation of the latest arrival times in a detour to node v_k from node v_i.

arrival times calculated for nodes v_i, v_k and v_j in the detour explained above. More formally, Algorithm 3 shows how these values are calculated in a preprocessing stage.

Algorithm 3. Latest arrival times calculation.

Input: $\mathcal{G}$, directed graph.
Output: **L**, latest arrival times.
for $(v_i, v_j) \in \mathcal{A}$ **do**
  for $v_k \in \mathcal{N}$ **do**
    $L^k_{(i,j)} \leftarrow -\infty$
    $L^k_{(k,j)} \leftarrow \min\{a_j - t_{k,j}, b_k\}$
    if $L^k_{(k,j)} \geq a_k$ `and` $\min\{L^k_{(k,j)} - t_{i,k}, b_i\} \geq a_i$ **then**
      $L^k_{(i,j)} \leftarrow \min\{L^k_{(k,j)} - t_{i,k}, b_i\}$
    end if
  end for
  end for
  return L

With these pre-calculated values, to prune a pulse that is trying to propagate from node v_i to v_j, the algorithm compares $t(\mathcal{P}_{s,i})$ against $L^k_{(i,j)}$ for every node $v_k \in \Gamma^+(v_i) \setminus \mathcal{P}_{s,i}$. Algorithm 4 shows the detour pruning strategy.

Algorithm 4. Pruning by detours.

Input: v_i, current node; v_j, destination node; $t(\mathcal{P}_{s,i})$ cumulative time; $\mathcal{P}_{s,i}$ partial path.
Output: `boolean`
`prune` $\leftarrow$ `false`
for $v_k \in \Gamma^+(v_i) \setminus \mathcal{P}_{s,i}$ **do**
  if $t(\mathcal{P}_{s,i}) \leq L^k_{(i,j)}$ **then**
    `prune` $\leftarrow$ `true`
  end if
end for
return `prune`

The idea behind this pruning strategy can be taken to a next level by demonstrating that arc $(v_i, v_j) \in \mathcal{A}$ can be removed from $\mathcal{G}$ if a detour from v_i to a given node v_k can take place within the time interval $[b_i, a_j]$. These arcs can be removed from the network at a preprocessing phase and need not to be considered in the exploration. The following theorem supports this preprocessing procedure.

Theorem 3.1. *For any arc $(v_i, v_j) \in \mathcal{A}$, if it is feasible to detour from v_i to a given node v_k within the time interval $[b_i, a_j]$, then $(v_i, v_j) \in \mathcal{A}$ can be removed from the graph $\mathcal{G}$ without sacrificing optimality.*

Proof. Let $\mathcal{P}^*$ be an optimal solution, $\mathcal{A}(\mathcal{P}^*)$ be the set of arcs in $\mathcal{P}^*$, and $\mathcal{N}(\mathcal{P}^*)$ be the set of nodes visited in $\mathcal{P}^*$. Under the assumption that $(v_i, v_j) \notin \mathcal{A}(\mathcal{P}^*)$, removing arc $(v_i, v_j) \in \mathcal{A}$ does not change $\mathcal{P}^*$ or the score $s(\mathcal{P}^*)$. Now assume that $(v_i, v_j) \in \mathcal{A}(\mathcal{P}^*)$. Let $\mathcal{P}^* = \mathcal{P}^*_{s,i} \cup \mathcal{P}^*_{j,e}$, where $\mathcal{P}^*_{s,i}$ is the path from v_s to v_i and $\mathcal{P}^*_{j,e}$ the path from v_j to v_e. Let $s(\mathcal{P}^*) = s(\mathcal{P}^*_{s,i}) + s(\mathcal{P}^*_{j,e})$ and $t(\mathcal{P}^*) = t(\mathcal{P}^*_{s,i}) + t_{i,j} + t(\mathcal{P}^*_{j,e})$. If $v_k \notin \mathcal{N}(\mathcal{P}^*)$, then $\mathcal{P}^*$ cannot be optimal since there is a feasible solution $\mathcal{P}' = \mathcal{P}^*_{s,i} \cup \{v_k\} \cup \mathcal{P}^*_{j,e}$ where $s(\mathcal{P}') = s(\mathcal{P}^*_{s,i}) + s_k + s(\mathcal{P}^*_{j,e}) > s(\mathcal{P}^*)$ with $t(\mathcal{P}') = t(\mathcal{P}^*_{s,i}) + t_{i,k} + t_{k,j} + t(\mathcal{P}^*_{j,e})$ and $t(\mathcal{P}^*_{s,i}) + t_{i,k} + t_{k,j} \leq a_j$. If $v_k \in \mathcal{N}(\mathcal{P}^*)$, then $v_k \in \mathcal{P}^*_{s,i}$ or $v_k \in \mathcal{P}^*_{j,e}$. If arc $(v_i, v_j) \in \mathcal{A}$ is removed, an alternative optimal solution $\mathcal{P}' = \mathcal{P}^*_{s,i} \setminus \{v_k\} \cup \{v_k\} \cup \mathcal{P}^*_{j,e}$ or $\mathcal{P}'' = \mathcal{P}^*_{s,i} \cup \{v_k\} \cup \mathcal{P}^*_{j,e} \setminus \{v_k\}$ exists. Notice that $\mathcal{N}(\mathcal{P}') = \mathcal{N}(\mathcal{P}'') = \mathcal{N}(\mathcal{P}^*)$, then $s(\mathcal{P}') = s(\mathcal{P}'') = s(\mathcal{P}^*)$ and either $\mathcal{P}'$ or $\mathcal{P}''$ are feasible because detouring to v_k is feasible within the time interval $[b_i, a_j]$. □

4. Parallelizing the algorithm

The pulse algorithm explores many partial paths before and after finding the optimal solution. In a sense, the paths that the algorithm explores are almost independent because the `pulse` function carries along the information of the path; only a binary array that keeps track of the visited nodes and the primal bound information have a global scope. For this reason, it is possible to perform a parallel search of the solution space taking care of how to update the global information. Once we obtain all the required bounds (see Section 3.1.2), we can invoke multiple times the `pulse` function using different parallel threads to accelerate the search. To do so, a fixed number of threads are triggered at node v_s propagating pulses through different outgoing arcs of v_s as proposed in Lozano and Medaglia [19]. Each thread uses a separate binary array for marking the visited nodes to avoid *race conditions*. Two threads can execute the `pulse` function over the same node at the same time except for the end node v_e, where threads must wait in queue for other threads to finish because the primal bound can be updated only by one thread at a time. Note that the number of threads determines the number of parallel explorations in progress; a new exploration is only triggered once a thread finishes its recursion stack. In the next section, we conduct a comprehensive set of experiments with different multi-thread versions of the algorithm to show the effect of parallelization in the performance of the algorithm.

5. Computational experiments

We tested our algorithm over the well-known Solomon's benchmark for the VRPTW [27], over the instances proposed by Cordeau et al. [7] for the Periodic Vehicle Routing Problem with Time Windows (PVRPTW), and over large-scale instances derived from Schilde et al. [26]. Solomon's testbed contains 29 instances with 100 nodes organized in three categories: the r-instances, where nodes are randomly located; the c-instances, where nodes are located in clusters; and the rc-instances, where some nodes are randomly located and others are clustered. Cordeau's testbed contains 10 instances that range from 48 to 288 nodes. These instances are considered to be harder because the time windows are wider, making the problems less constrained. The testbed from Schilde et al. [26] comprises real world networks ranging from 92 to 2143 nodes. Solomon's testbed is available at http://w.cba.neu.edu/~msolomon/problems.htm and Cordeau's testbed is available at http://neumann.hec.ca/chairedistributique/data/.

We coded our algorithm in Java, using Eclipse SDK version 4.2.1, and performed our experiments on a computer with an Intel Core i7-3517U @1.9 GHz CPU (2 cores) with 512 MB of RAM allocated to the memory heap size of the Java Virtual Machine on Windows 8. After tuning the algorithm parameters, we set the time step Δ to 5 and the time threshold $\underline{t}$ to 30% of the global time constraint T. We reported the computational time with a precision of 1/100 s; any time under 0.005 is reported as 0.00.

5.1. Comparison against the state-of-the-art algorithm

For the OPTW, the state-of-the-art algorithm is a bi-directional and bounded DP algorithm with DSSR by Righini and Salani [24]. They coded their algorithm in ANSI-C compiled with gcc 3.0.4 and executed their experiments on a Pentium IV 1.6 GHz processor with 512 MB RAM with a time limit of 2 h. For the sake of fairness, we scaled our times by 4.3 according to the LINPACK benchmark [9].

To adapt instances from the VRPTW to the OPTW, we replicated the same approach used by Righini and Salani [24] in which they consider the demand at each node v_i in the original Solomon's and

Cordeau's instances as the score s_i. Tables 1 and 2 present a head-to-head comparison between the 4-thread version of our pulse algorithm and the best approach for each instance presented by Righini and Salani [24]. Column 1 shows the instance name; column 2 presents the optimal score; column 3 shows the time in seconds as reported by Righini and Salani [24]; column 4 displays the scaled computational time of our algorithm; and finally, column 5 shows the corresponding speedup for each instance calculated as the ratio of column 3 over column 4. At the end of the table we present the arithmetic and geometric mean of the speedups. Note that the geometric mean provides a fairer comparison between the algorithms since large ratios on few instances do not affect sharply the measure as it often happens with the arithmetic mean [4].

Table 1 shows that in 28 out of 29 instances the pulse outperforms the state-of-the-art algorithm, achieving speedups of up to 266. On average, the arithmetic mean of the speedup shows that the pulse algorithm is 70 times faster, whereas the more conservative geometric mean exhibits an average speedup of 27 times. Some instances that are not solved by the DSSR algorithm in a time limit of 2 h are successfully solved by the pulse algorithm in less than 80 s. It is also noteworthy that 27 out of 29 instances are solved in less than a minute and all the instances are solved in less than two minutes.

Table 2 presents the same performance comparison made over the harder Cordeau's testbed. In summary, the pulse algorithm also outperforms DSSR in Cordeau's testbed. The speedups over this testbed range from 8 to 186 times. The average speedups

Table 1
Head-to-head comparison of the pulse algorithm against the state-of-the-art algorithm for the OPTW over Solomon's testbed. Bold values indicate the faster algorithm.

Instance	Optimal score	Righini and Salani DSSR (s)	Pulse (s)	Speedup
C101_100	320	0.06	**0.00**	13.85
C102_100	360	3.81	**0.54**	7.03
C103_100	400	1081.04	**8.94**	120.87
C104_100	420	1856.39	**8.94**	207.56
C105_100	340	0.12	**0.07**	1.85
C106_100	340	0.14	**0.07**	2.02
C107_100	370	0.20	**0.07**	2.88
C108_100	370	1.43	**0.14**	10.31
C109_100	380	10.57	**0.34**	31.27
R101_100	198	**0.03**	0.07	0.43
R102_100	286	233.20	**7.51**	31.04
R103_100	293	5498.81	**31.29**	175.76
R104_100	303	> 7200	**80.04**	> 89.95
R105_100	247	0.23	**0.07**	3.54
R106_100	293	334.49	**12.66**	26.42
R107_100	299	2979.94	**35.49**	83.98
R108_100	308	> 7200	**75.64**	> 95.18
R109_100	277	3.09	**0.20**	15.17
R110_100	284	30.83	**0.75**	41.36
R111_100	297	1408.80	**14.56**	96.79
R112_100	298	2508.17	**9.41**	266.49
RC101_100	219	0.23	**0.07**	3.54
RC102_100	266	6.11	**0.13**	45.48
RC103_100	266	88.12	**0.47**	186.56
RC104_100	301	264.84	**1.56**	170.24
RC105_100	244	2.86	**0.07**	44.00
RC106_100	252	2.08	**0.13**	15.48
RC107_100	277	49.19	**0.41**	120.76
RC108_100	298	68.95	**0.81**	85.09
Arithmetic mean				68.79
Geometric mean				27.83

Table 2
Head-to-head comparison of the pulse algorithm against the state-of-the-art algorithm for the OPTW over Cordeau's testbed. Bold values indicate the faster algorithm.

Instance	Optimal score	Righini and Salani DSSR (s)	Pulse (s)	Speedup
pr01_48	308	1.19	**0.14**	8.58
pr02_96	404	37.52	**1.22**	30.81
pr03_144	394	151.73	**2.30**	65.94
pr04_192	489	648.82	**4.06**	159.62
pr05_240	595	6815.82	**36.57**	186.36
pr06_288	591	> 7200	**78.23**	> 92.04
pr07_72	298	3.65	**0.27**	13.59
pr08_144	463	90.71	**1.69**	53.67
pr09_216	493	3270.88	**81.01**	40.38
pr10_288	594	> 7200	**64.26**	> 112.04
Arithmetic mean				76.30
Geometric mean				52.45

reassert that the pulse algorithm is 76 and 52 times faster than DSSR with regard to the arithmetic and the geometric mean, respectively. Note that the pulse algorithm consistently solves larger instances (of up to 288 nodes) with wider time windows in less than 2 min.

5.2. Experiments on large-scale networks

We tested the proposed pulse algorithm over large-scale instances derived from the work of Schilde et al. [26] on the multiobjective orienteering problem (MOOP), where each node has two scores (objectives). We considered networks ranging from 273 to 562 nodes and created OPTW instances by dropping the second objective and generating time windows for every node following a procedure similar to that outlined by Solomon [27]. For each network, we generated 12 instances varying the maximum travel time T and the tightness of the time windows. To vary the maximum travel time, we defined $\overline{T}$ as the maximum T used by Schilde et al. [26] for each network and built instances with T equal to roughly 25%, 50%, 75%, and 100% of $\overline{T}$. For each value of T, we generated three variants changing the tightness of the time windows. To do so, we first generate the centroid of the time window for each node v_i following a uniform distribution in the interval defined by the earliest possible arrival time to v_i and the latest possible departure time, i.e., $[t_{s,i}, T - t_{i,e}]$. With respect to these random centroids, we set the width of the time windows as follows: tight (T) time windows have a width uniformly distributed between $[0.10T, 0.15T]$, i.e., between 10% and 15% of the maximum travel time T; medium (M) time windows have a width uniformly distributed between $[0.15T, 0.20T]$; and loose (L) time windows have a width uniformly distributed between $[0.20T, 0.25T]$. It is important to mention that on this testbed, the triangle inequality does not hold, which means that for this experiment our algorithm was unable to use the detour pruning strategy and the arc deletion preprocessing procedure.

Table 3 presents the computational results for the 36 generated instances. Column 1 presents the number of nodes in the network; column 2 shows the maximum travel time for each instance; column 3 presents the tightness of the time windows; column 4 shows the optimal value of the objective function; and finally, column 5 reports the computational time (in seconds).

On these large-scale networks, the pulse algorithm was able to solve all instances within one hour, but as the windows become loose, the harder they become. On instances with 273 nodes, it solved all problems in less than 12 s. For instances with 559 nodes, the algorithm solves the tighter instances in a couple of milliseconds, but as the time windows become loose, the computational time increases up to 1713 s. Similarly, for the instances with 562 nodes the algorithm solved 6 out of 12 instances in less than one second, but the harder problem took up to 60 s. This experiment shows that as the time constraints are relaxed, the computational time required by the pulse algorithm to solve the problems grows. Moreover, the computational time seems to be more sensitive to changes in the tightness of the time windows. This may be explained by the fact that with wider time windows the algorithm needs to calculate more bounds in the bounding procedure (see Section 3.1.2). Nevertheless, it is noteworthy that even though these instances are up to five times larger than the instances in Solomon's testbed (and twice those of Cordeau), the algorithm was still able to efficiently solve these problems.

Table 3
Computational results for the large-scale instances derived from Schilde et al. [26].

Number of nodes	T	Windows tightness	Optimal score	Pulse time (s)
273	5	T	280	0.02
		M	327	0.03
		L	358	0.03
	10	T	688	0.08
		M	728	0.13
		L	746	0.23
	15	T	1156	0.33
		M	1213	0.66
		L	1226	1.52
	20	T	1387	0.97
		M	1508	2.13
		L	1581	11.94
559	40	T	241	0.06
		M	284	0.06
		L	284	0.06
	75	T	635	0.16
		M	716	0.22
		L	716	0.33
	110	T	1092	3.14
		M	1213	41.33
		L	1260	901.89
	150	T	1507	6.59
		M	1649	62.28
		L	1768	1713.38
562	40	T	562	0.09
		M	574	0.11
		L	603	0.13
	75	T	1086	0.31
		M	1242	0.81
		L	1293	0.89
	110	T	1386	1.75
		M	1494	4.19
		L	1673	11.82
	150	T	1674	7.72
		M	1756	44.71
		L	1978	60.12

5.3. Assessing the strength of the problem-specific pruning strategies

To measure the strength of the proposed problem-specific strategies, we compared two versions of the pulse algorithm: a *basic pulse algorithm* that only uses the core pruning strategies; and an *enhanced pulse algorithm* that relies on both the core and problem-specific pruning strategies. For the comparison we chose the hardest instances from Solomon's and Cordeau's testbeds (i.e., those that took more than one second to solve), and we measured the computational time and the number of pulses that reached the end node. Note that if a large number of pulses reach the end node it means that the pruning strategies are weak and they are not pruning pulses efficiently. On the contrary, when few pulses reach the end node the metric is a good proxy of the strength of the pruning strategies. To eliminate confounding factors, both versions of the algorithm were tested using a single thread of execution. Table 4 presents the results of this experiment. Column 1 shows the name of the instance; columns 2 and 3 present the computational time in seconds and the amount of pulses reaching the end node with the basic pulse algorithm; columns 4 and 5 show the same performance measures but for the enhanced pulse algorithm; column 6 presents the speedup achieved by the enhanced version of the algorithm; and finally, column 7 presents the exploration fraction calculated as the number of paths explored

Table 4
Assessing the strength of the problem-specific pruning strategies.

Instance	Core pulse		Enhanced pulse		Speedup	Exploration fraction
	Time (s)	Paths	Time (s)	Paths		
C103_100	3.36	129,850	2.24	83,016	1.50	0.64
C104_100	3.59	198,857	2.32	111,419	1.55	0.56
R102_100	5.08	619,111	1.87	136,527	2.72	0.22
R103_100	38.33	9,464,148	10.36	1,873,775	3.70	0.20
R104_100	120.92	26,679,302	27.13	4,492,992	4.46	0.17
R106_100	8.75	1,435,291	3.20	330,081	2.74	0.23
R107_100	42.47	10,855,293	12.05	2,242,278	3.53	0.21
R108_100	107.96	22,539,854	25.71	3,955,766	4.20	0.18
R111_100	8.38	1,427,060	3.55	384,245	2.36	0.27
R112_100	6.83	1,196,465	2.88	356,979	2.38	0.30
pr04_192	2.14	214,451	1.36	111,366	1.58	0.52
pr05_240	18.25	771,485	9.92	354,697	1.84	0.46
pr06_288	50.74	279,239	19.45	158,610	2.61	0.57
pr09_216	51.97	434,716	19.47	158,243	2.67	0.36
pr10_288	31.55	654,757	16.32	330,972	1.93	0.51
Arithmetic mean					2.65	0.36

Table 5
Evaluating the effect of the number of threads on the performance of the pulse algorithm.

Instances	Time for τ number of threads (s)					Speedup
	1	2	4	8	16	
C103_100	2.24	2.09	2.06	1.98	1.95	1.15
C104_100	2.32	2.17	2.06	2.05	2.00	1.16
R102_100	1.87	1.73	1.73	1.78	1.75	1.08
R103_100	10.36	7.99	7.22	7.47	7.49	1.43
R104_100	27.13	20.88	18.47	19.13	19.57	1.47
R106_100	3.20	2.88	2.92	2.97	3.00	1.11
R107_100	12.05	8.55	8.19	8.17	8.18	1.47
R108_100	25.71	18.95	17.46	17.93	18.32	1.47
R111_100	3.55	3.33	3.36	3.05	3.03	1.17
R112_100	2.88	2.91	2.17	2.25	2.03	1.41
pr04_192	1.36	1.16	0.94	0.94	0.95	1.45
pr05_240	9.92	8.91	8.44	8.75	8.31	1.19
pr06_288	19.45	18.13	18.05	20.00	17.41	1.12
pr09_216	19.47	18.08	18.69	20.58	18.08	1.08
pr10_288	16.32	14.91	14.83	14.97	15.99	1.10
Arithmetic mean	10.52	8.84	8.44	8.80	8.54	1.26

in the enhanced pulse over the number of paths explored in the basic version.

As expected, by the design of the experiment, the enhanced pulse clearly outperforms the basic pulse across all instances. The problem-specific strategies in the enhanced version are responsible for pruning at early stages a huge number of paths that lead to average speedups of 2.65. The number of paths that reach the end node in the enhanced version was reduced from the basic version on average by 64%. Moreover, the r-instances show the largest reduction in time, where the explored paths are reduced by roughly 80% on the hardest instances of Solomon's testbed (R103, R104, R107, and R108). In this sense, we show that the proposed problem-specific pruning strategies improve the performance of the pulse algorithm leading to significant reductions in computational time and in the number of complete solutions explored.

5.4. Effect of the parallelization

We parallelized the enhanced pulse as explained in Section 4. In this experiment, we used the same set of hard instances from Section 5.3 and evaluated the parallel version of the pulse algorithm with 2, 4, 8, and 16 threads. Table 5 summarizes the results. Column 1 shows the name of the instance; columns 2–6 show the computational time of the algorithm varying the number of threads; column 7 shows the speedup calculated as the ratio between the time of the single-thread version over the best time obtained by the multi-thread versions.

Table 5 shows a clear reduction in computational time when more than one thread is used. Speedups range from 1.08 to 1.47, and on average, the multi-thread version is approximately 26% faster than the single-thread version. The computational time stabilizes when four threads or more are used, which is expected given the CPU (2 cores and 4 threads) used in the experiment. These speedups might be the result of the multiple paths explored in parallel, obtaining better primal bounds in earlier stages of the algorithm, which in turn, strengthens the bounds pruning strategy.

6. Concluding remarks

We outlined how the pulse algorithm [19,18] can be seen as a framework for hard shortest path variants which are at the core of the solution of hard routing problems. In this paper, we successfully adapted the pulse algorithm for the OPTW by adding new problem-specific strategies that exploit the problem structure. Extensive computational experiments showed that our algorithm outperforms the state-of-the-art algorithm by Righini and Salani [24] on two different benchmark instances from the literature, achieving speedups of up to 266 in Solomon's dataset and 186 in Cordeau's dataset. On average, our algorithm exhibited a 70-fold speedup (arithmetic mean) over both datasets. Moreover, using the more conservative geometric mean, we found speedups of 27 and 52 for Solomon's and Cordeau's datasets, respectively.

To stress the algorithm, we also conducted a set of computational experiments over a testbed derived from Schilde et al. [26], with instances ranging from 273 to 562 nodes. Over these large-scale problems, the pulse algorithm was able to solve all instances within 1700 s. We observed that computation time is highly dependent on the width of the time windows. As the time constraints become loose, the problem becomes harder for the pulse algorithm because it needs to calculate more bounds and the implicit enumeration needs to deal with a solution space that grows in size.

We assessed the strength of the new problem-specific pruning strategies. These new pruning strategies are a key factor in the performance of the algorithm, achieving speedups of up to 4.46 times. On average, problem-specific pruning strategies accelerated the basic pulse 2.65 times and reduced the explored paths by 64%.

We also explored the effect of the parallelization of the algorithm on the computational time. The multi-thread version of the algorithm speeds up by 26% (on average) the single-thread version; and multi-threading accelerates the algorithm up to 47% in certain instances. In summary, the parallelization of the pulse algorithm showed potential for scalability and improved performance, without being overly expensive in terms of the coding effort.

Research currently underway aims to adapt the pulse algorithm to stochastic variants of the OPTW problem and other multi-constrained variants of the OP. We would like to embed the algorithm as a building-block in large-scale solution schemes (e.g., column generation procedures) to tackle large and hard routing problems. Finally, heuristic versions based on the pulse framework might be useful to obtain high quality solutions for the OPTW and other variants.

References

[1] Archetti C, Bianchessi N, Speranza MG. Optimal solutions for routing problems with profits. Discrete Appl Math 2013;161:547–57.
[2] Archetti C, Bianchessi N, Speranza MG, Hertz A. Incomplete service and split deliveries in a routing problem with profits. Networks 2014;63(2):135–45.

[3] Archetti C, Bianchessi N, Speranza MG, Hertz A. The split delivery capacitated team orienteering problem. Networks 2014;63(1):16–33.
[4] Bixby RE. Solving real-world linear programs: a decade and more of progress. Oper Res 2002;50(1):3–15.
[5] Bolívar MA, Lozano L, Medaglia AL. Acceleration strategies for the weight constrained shortest path problem with replenishment. Optim Lett. Berlin, Heidelberg: Springer; 2014:1–8. http://dx.doi.org/10.1007/s11590-014-0742-x.
[6] Chao I, Golden BL, Wasil EA. A fast and effective heuristic for the orienteering problem. Eur J Oper Res 1996;88(3):475–89.
[7] Cordeau J-F, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30(2):105–19.
[8] Desrochers M, Desrosiers J, Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 1992;40(2):342–54.
[9] Dongarra JJ. Performance of various computers using standard linear equations software. Technical report CS-89-85. USA: University of Tennessee; 2009.
[10] Duque D, Lozano L, Medaglia AL. An exact method for the biobjective shortest path problem for large-scale networks. Eur J Oper Res 2014. In press. http://dx.doi.org/10.1016/j.ejor.2014.11.003.
[11] Feillet D, Dejax P, Gendreau M, Gueguen C. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 2004;44(3):216–29.
[12] Fischetti M, Salazar JJ, Toth P. Solving the orienteering problem through branch-and-cut. INFORMS J Comput 1998;10(2):133–48.
[13] Gambardella LM, Montemanni R, Weyland D. Coupling ant colony systems with strong local searches. Eur J Oper Res 2012;220(3):831–43.
[14] Gendreau M, Laporte G, Semet F. A branch-and-cut algorithm for the undirected selective travelling salesman problem. Networks 1998;32(4):263–73.
[15] Gendreau M, Laporte G, Semet F. A tabu search heuristic for the undirected selective travelling salesman problem. Eur J Oper Res 1998;106:539–45.
[16] Golden BL, Levy L, Vohra R. The orienteering problem. Naval Res Logist 1987;34(3):307–18.
[17] Ílhan T, Iravani SMR, Daskin MS. The orienteering problem with stochastic profits. IIE Trans 2008;40(4):406–21.
[18] Lozano L, Duque D, Medaglia AL. An exact algorithm for the elementary shortest path problem with resource constraints. Transp Sci 2014. Available at: http://hdl.handle.net/1992/1181.
[19] Lozano L, Medaglia AL. On an exact method for the constrained shortest path problem. Comput Oper Res 2013;40(1):378–84.
[20] Montemanni R, Gambardella LM. An ant colony system for team orienteering problems with time windows. Found Comput Decis Sci 2009;34(4):287–306.
[21] Moser HD. Scheduling and routing tactical aerial reconninssance vehicles [Master 's thesis]. Naval Postgraduate School; 1990.
[22] Ramesh R, Brown KM. An efficient four-phase heuristic for the generalized orienteering problem. Comput Oper Res 1991;18(2):151–65.
[23] Righini G, Salani M. New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks 2008;51(3):155–70.
[24] Righini G, Salani M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Comput Oper Res 2009;36(4):1191–203.
[25] Royset JO, Reber DN. Optimized routing of unmanned aerial systems for the interdiction of improvised explosive devices. Mil Oper Res 2009;14(4):5–19.
[26] Schilde M, Doerner KF, Hartl RF, Kiechle G. Metaheuristics for the biobjective orienteering problem. Swarm Intell 2009;3(3):179–201.
[27] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time windows constraints. Oper Res 1987;35(2):254–65.
[28] Souffriau W, Vansteenwegen P, Vertommen J, Vanden Berghe G, Van Oudheusden D. A personalized tourist trip design algorithm for mobile tourist guides. Appl Artif Intell 2008;22(10):964–85.
[29] Tang H, Miller-Hooks E. A tabu search heuristic for the team orienteering problem. Comput Oper Res 2005;32(6):1379–407.
[30] Tang H, Miller-Hooks E, Tomastik R. Scheduling technicians for planned maintenance of geographically distributed equipment. Transp Res Part E: Logist Transp Rev 2007;43(5):591–609.
[31] Tricoire F, Romauch M, Doerner KF, Hartl RF. Heuristics for the multi-period orienteering problem with multiple time windows. Comput Oper Res 2010;37(2):351–67.
[32] Tsiligirides T. Heuristic methods applied to orienteering. J Oper Res Soc 1984;35(9):797–809.
[33] Vansteenwegen P, Souffriau W, Van Oudheusden D. The orienteering problem: a survey. Eur J Oper Res 2011;209(1):1–10.
[34] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. Iterated local search for the team orienteering problem with time windows. Comput Oper Res 2009;36(12):3281–90.
[35] Verbeeck C, Sörensen K, Aghezzaf E-H, Vansteenwegen P. A fast solution method for the time-dependent orienteering problem. Eur J Oper Res 2014;236(2):419–32.
[36] Wang X, Golden BL, Wasil EA. Using a genetic algorithm to solve the generalized orienteering problem. In: Golden B, Raghavan S, Wasil EA, editors. The vehicle routing problem: latest advances and new challenges. Operations research/computer science interfaces. New York, US: Springer; 2008.