Open access • Journal Article • DOI:10.1007/S11590-014-0742-X

# Acceleration strategies for the weight constrained shortest path problem with replenishment — Source link

Manuel A. Bolívar, Leonardo Lozano, Andrés L. Medaglia

**Institutions:** University of Los Andes

**Published on:** 17 Apr 2014 - Optimization Letters (Springer Berlin Heidelberg)

**Topics:** K shortest path routing, Constrained Shortest Path First, Shortest path problem, Shortest Path Faster Algorithm and Yen's algorithm

# Acceleration strategies for the Weight Constrained Shortest Path Problem with Replenishment

Manuel A. Bolívar, Leonardo Lozano, Andrés L. Medaglia[1]

*Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de Los Andes, Bogotá, Colombia.*

## Abstract

The Weight Constrained Shortest Path Problem with Replenishment (WCSPP-R) generalizes the Constrained Shortest Path Problem (CSP) and has multiple applications in transportation, scheduling, and telecommunications. We present an exact algorithm that combines and extends the ideas proposed in the state-of-the-art algorithms for the CSP and WCSPP-R. The novelty lies in a set of acceleration strategies that significantly improves the algorithm's performance. We conducted experiments over large real-road networks achieving speedups up to 219 against the state-of the-art algorithm.

**Keywords**: Shortest path problem, replenishment, resource constraint, large-scale networks.

### *1. Introduction*

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph defined by a set $\mathcal{N} = \{v_1, \dots, v_i, \dots, v_n\}$ of nodes and a set $\mathcal{A} = \{(i,j) | v_i \in \mathcal{N}, v_j \in \mathcal{N}, i \neq j\}$ of directed arcs. Each arc $(i,j) \in \mathcal{A}$ has an associated cost $c_{ij}$, a resource consumption $w_{ij}$, and a binary indicator $r_{ij}$ that takes the value of 1 if the arc is a replenishment arc (it takes the value of 0, otherwise). The Weight Constrained Shortest Path Problem with Replenishment (WCSPP-R) consists of finding the minimum cost path between a start node $v_s \in \mathcal{N}$ and an end node $v_t \in \mathcal{N}$ without exceeding a resource constraint $W$ while taking into account replenishment arcs that reset the value of the consumed resource to zero (at the start of the arc).

Replenishment opportunities arise naturally in several contexts related to shortest paths. For instance, in airline crew pairing the rest periods replenish the workability of a crew (Azadeh et al., 2013); equivalently, in aircraft routing, preventive maintenance allows the aircraft to keep flying (Wang & Pham, 2013). In vehicle routing, fuel or energy replenishment may be necessary to reach the final destination. A clear example arises in the recent context of electric vehicles (EVs). EVs have seen a tremendous growth in recent years due to their low carbon emission, the high price of fuel, and public environmental concerns (Gallardo-Lozano, 2012). However, current EVs have some major disadvantages, namely, limited autonomy and long charging times (Kobayashi, 2011). Although some of these problems have been dealt with computer-aided intelligent routing (Lee &

[1] Corresponding author. Universidad de Los Andes, Cr 1E No. 19A-10, ML711, Bogotá, Colombia. Tel: +57 (1) 3393949x2880; e-mail: amedagli@uniandes.edu.co; URL: http://wwwprof.uniandes.edu.co/~amedagli

Park, 2013), the literature has seldom addressed the inclusion of these replenishment opportunities.

The WCSPP-R has not been studied, with the notable exception of Smith et al. (2012) who proposed two exact algorithms. The first one uses a meta-network (or high level network) to exploit the inter-replenishment subpath structure of feasible paths. The second one is a label correcting algorithm combined with an efficient preprocessing technique used to aggressively reduce the size of the network, removing nodes and arcs by the use of primal and dual bounds. They tested the algorithms over two types of networks: the first one, a set of randomly generated grid networks; and the second one, a set of acyclic networks that arise as subproblems in a branch-and-price approach for an airline crew pairing problem. In these computational experiments, the label correcting algorithm outperforms the method based on the meta-network.

A problem related to the WCSPP-R is the shortest path problem with relays (SPPR). This problem consists of finding the minimum cost path from an origin to a destination, subject to a resource constraint, and using nodes that reset the resource consumption according to a node dependent cost attribute. Cabral et al. (2005) introduced three solution methods for this problem and later, Cabral et al. (2008) used the SPPR as a subproblem for a network design problem. More recently, Laporte & Pascoal (2011) proposed a new strategy to solve the SPPR based on a label correcting algorithm; additionally, they solved a variant of the problem where the path and relay costs are considered separately.

Finally, for the CSP there is plenty of literature available. Joksch (1966) proposed a dynamic programming algorithm to solve the CSP, lately extended by Dumitrescu & Boland (2003) by including preprocessing technics. Handler & Zang (1980) use a $k$-shortest path algorithm where they identify $k$ paths, sort them by length and evaluate them successively until finding the first feasible path. Santos et al. (2007) extended this idea by improving the search direction based on the relative tightness of the resource constraint. More recently, Lozano & Medaglia (2013) proposed an exact algorithm using a recursive depth-first search exploration combined with pruning strategies to avoid a complete exploration of the network.

The contribution of this paper is twofold: from a methodological perspective, we present a set of acceleration strategies that combines depth and breadth search, generalizing the ideas proposed by Smith et al. (2012) and Lozano & Medaglia (2013), both state-of-the-art algorithms for the WCSPP-R and CSP, respectively. From a computational perspective, we provide a vast set of real-road networks for the WCSPP-R and conduct extensive computational experiments achieving remarkable speedups of up to 219 times against the state-of-the-art algorithm for the WCSPP-R.

The remainder of this paper is organized as follows: Section 2 presents an overview of the algorithms proposed by Smith et al. (2012) and Lozano & Medaglia (2013), and outlines the intuition behind the acceleration strategies. Section 3 provides a detailed description of the acceleration strategies. Section 4 presents the computational results. Finally, Section 5 concludes the paper and outlines future work.

### *2. Previous work and acceleration strategies intuition*

We extend the Label Correcting (LC) algorithm proposed by Smith et al. (2012) for the WCSPP-R and the *pulse algorithm* presented by Lozano & Medaglia (2013) for the CSP. Both approaches are based on the idea of exploiting the underlying information of the network (e.g., primal and dual bounds) with the purpose of avoiding a complete exploration of the graph without jeopardizing the optimal solution. In this section, we overview the LC and pulse algorithms in order to introduce our approach and present the acceleration strategies intuition.

The Label Correcting (LC) algorithm (Smith et al., 2012) comprises two stages. The first stage is a preprocessing procedure that finds dual (lower) and primal (upper) bounds on the cost and weight incurred from $v_s$ to any node $v_i \in \mathcal{N}$ and from every node $v_i \in \mathcal{N}$ to the sink node $v_t$ . Then it aggressively removes from the network those nodes and arcs that cannot be part of the optimal solution or cannot improve the primal bound. A second stage comprises an implicit complete graph exploration via a LC algorithm where the labels are processed with a given treatment criterion. Each label in a node $v_i$ represents a partial path $\mathcal{P}_{si}$ from $v_s$ to $v_i$ and consists of three elements: the node $v_i$, the cumulative cost $c(\mathcal{P}_{si})$, and the consumed resource $w(\mathcal{P}_{si})$. The LC algorithm starts with an empty set of untreated labels that is triggered by setting a label in $v_s$ with cost and resource consumption equal to zero. According to the label treatment criterion, labels are pulled out from the untreated label set and extended from their node along each outgoing arc, generating new labels. However, not all the outgoing arcs generate new labels because infeasible and dominated labels are discarded. The LC algorithm stops when the untreated label set is empty.

Equivalently, the pulse algorithm (Lozano and Medaglia, 2013) also comprises two stages: 1) a bounding stage that finds dual bounds on the cost and the resource consumption from any node $v_i$ to the final node $v_t$, and 2) a recursive exploration stage that finds the optimal solution based on an implicit enumeration of the solution space. The exploration is started by sending a pulse from the start node $v_s$. The pulse tries to propagate throughout the outgoing arcs of each visited node recursively, storing at each node the partial path $\mathcal{P}$, the cumulative cost $c(\mathcal{P})$ and the cumulative resource consumption $w(\mathcal{P})$ . At each node, different pruning strategies try to prevent pulse propagation based on partial path dominance, infeasibility, and bounds. Every pulse that reaches the sink node $v_t$ contains all the information of a feasible path from $v_s$ to $v_t$ and possibly updates the global primal bound.

It is worth noting the similarities between these parallel research lines. First, both algorithms find cost and resource bounds for all nodes. Nonetheless, the preprocessing of the LC algorithm goes a step further and uses the computed information to remove nodes and arcs from the graph. Second, although the exploration order may differ from each other, the ideas of discarding a label or pruning a pulse are very similar. Finally, both algorithms use analogous strategies to prevent the propagation of labels and pulses, i.e., infeasibility, dominance, and bounds.

However, the approaches present important differences in the way the graph is explored. On one hand, the LC algorithm explores all the successors of a label's node and then globally selects the

next label to be extended according to the label treatment criterion following a lexicographic breadth-first search (Corneil, 2005). On the other hand, the pulse algorithm follows a pure depth-first search because a pulse propagates recursively until it is pruned or reaches the sink node. Finally, the pulse algorithm updates the incumbent primal bound when the sink node $v_t$ is reached; whereas the LC algorithm does not update the primal bound.

Our proposed approach puts together the best from both algorithms under the pulse framework. In a first stage it obtains primal and dual bounds using the flawless preprocessing procedure by Smith et al. (2012). In a second stage, it explores the network using a modified pulse algorithm with new pruning strategies and an enhanced exploration order. Aside from the basic pruning strategies, i.e., infeasibility, dominance, and bounds, we prune pulses based on a *path completion* strategy. Additionally, when the depth of a pulse $d(\rho)$ reaches a maximum allowed value $\delta$, it is paused and stored in a *pulse queue* $\Sigma$. Each paused pulse $\rho$ is defined as a tuple: $(n(\rho), c(\rho), w(\rho), d(\rho), \mathcal{P})$ where $n(\rho)$ is the node where the pulse was paused while $c(\rho), w(\rho), d(\rho)$ are the cumulative cost, resource, and depth, and $\mathcal{P}$ is the partial path of the paused pulse. The algorithm stops when the pulse queue is empty. It is noteworthy that the way the algorithm explores the graph depends on the value of $\delta$: If $\delta$ is equal zero, the exploration becomes a lexicographic breadth-first search; on the other hand, if $\delta$ is large enough, the exploration becomes depth-first search. Intermediate values for $\delta$ will combine depth-first with breadth-first search strategies.

Algorithm 1 presents the pseudocode of the proposed algorithm. Lines 1 through 5 initialize the partial path $\mathcal{P}$, the pulse queue $\Sigma$, the initial cumulative cost $c^0$, the initial resource consumption $w^0$ and the initial depth $d^0$. Line 6 executes the preprocessing algorithm proposed by Smith et al. (2012). Line 7 starts the pulse queue $\Sigma$ with a paused pulse in node $v_s$. Lines 8 through 13 propagate pulses according to the queueing discipline. Line 9 extracts the next paused pulse to be processed and line 10 removes it from the pulse queue. Line 11 checks if the paused pulse can be discarded by bounds pruning. If the pulse is not discarded, line 12 propagates the pulse resuming at node $n(\rho)$. Finally, line 15 returns the optimal path found in the pulse function.

**Algorithm 1**. Pulse algorithm

**Input**: $\mathcal{N}, \mathcal{A}, W, v_s, v_t$
**Output**: Optimal path $\mathcal{P}^*$

1: $\mathcal{P} \leftarrow \{\,\}$
2: $\Sigma \leftarrow \{\,\}$
3: $c^0 \leftarrow 0$
4: $w^0 \leftarrow 0$
5: $d^0 \leftarrow 0$
6: $Preprocessing\ (\mathcal{N}, \mathcal{A}, W, v_s, v_t)$
7: $\Sigma \leftarrow \Sigma \cup \{(v_s, c^0, w^0, d^0, \mathcal{P})\}$
8: **while** $\Sigma$ is not empty **do**
9: $\quad \rho \leftarrow \text{nextPulse}(\Sigma)$
10: $\quad \Sigma \leftarrow \Sigma \setminus \{\rho\}$
11: $\quad$ **if** checkBounds$\big(n(\rho),\ c(\rho)\big)$ **then**

12: pulse$(n(\rho), c(\rho), w(\rho), d(\rho), \boldsymbol{\mathcal{P}})$
13: **end if**
14: **end while**
15: **return** $\boldsymbol{\mathcal{P}}^*$

Algorithm 2 shows the body of the recursive function *pulse* where $N(v_i) = \{v_j \in \mathcal{N} | (i,j) \in \mathcal{A}\}$ is the set of adjacent nodes from node $v_i$. Lines 2 through 5 update the cumulative cost, cumulative consumed resource, depth, and partial path of the pulse. Line 6 checks if it is feasible to reach the sink node $v_t$ from node $v_j$. Line 7 tries to prune the new pulse using bounds while line 8 checks if the partial path $\mathcal{P}'$ is dominated. Line 9 explores the possibility of completing the partial path $\mathcal{P}'$ with the minimum cost feasible path or updating the primal bound with the minimum-weight feasible path from node $v_j$ to the sink node $v_t$. Line 10 checks if the pulse has reached the maximum depth $\delta$, if so, line 11 adds the pulse $(v_j, c', w', 0, \mathcal{P}')$ to the pulse queue $\Sigma$ resetting its depth to zero. Finally, line 13 recursively propagates the pulse through node $v_j$.

**Algorithm 2**. *pulse* function
**Input**: $v_i, c, w, d, \mathcal{P}$
**Output**: void
1: **for** $v_j \in N(v_i)$ **do**
2: $c' \leftarrow c + c_{ij}$
3: $w' \leftarrow w \cdot (1 - r_{ij}) + w_{ij}$
4: $d' \leftarrow d + 1$
5: $\mathcal{P}' \leftarrow \mathcal{P} \cup \{v_j\}$
6: **if** checkFeasibility$(v_j, w') = true$ **then**
7: **if** checkBounds$(v_j, c') = false$ **then**
8: **if** checkDominace$(v_j, c', w') = false$ **then**
9: **if** checkCompletePath$(v_j, c', w') = false$ **then**
10: **if** checkDepth$(d') = true$ **then**
11: $\Sigma \leftarrow \Sigma \cup \{(v_j, c', w', 0, \mathcal{P}')\}$
12: **else**
13: pulse$(v_j, c', w', d', \mathcal{P}')$
14: **end if**
15: **end if**
16: **end if**
17: **end if**
18: **end if**
19: **end for**

Every time the pulse function is called over the sink node $v_t$ , it checks if the incumbent primal bound can be updated and the pulse propagation stops.

### *3. Acceleration strategies*

In addition to the pruning strategies for the CSP (Lozano & Medaglia, 2013), our algorithm uses three new acceleration strategies, namely, *path completion*, *pulse queueing,* and an *enhanced exploration* order. Henceforth we denote the minimum cost path from a node $v_i \in \mathcal{N}$ to the sink node $v_t$ by $\mathcal{P}_{it}^c$ and the minimum weight feasible path from $v_i$ to $v_t$ by $\mathcal{P}_{it}^w$. It is worth highlighting

that all minimum cost paths and minimum weight feasible paths from any node $v_i$ to $v_t$ are computed beforehand, in a preprocessing stage.

*3.1 Basic pruning strategies*

The basic pruning strategies used in both the LC algorithm and the pulse algorithm are: infeasibility, bounds, and dominance pruning. The infeasibility pruning strategy discards a partial path $\mathcal{P}_{si}$ when it already exceeds the resource constraint when it is completed with the minimum weight feasible path, i.e., $w(\mathcal{P}_{si}) + w(\mathcal{P}_{it}^{w}) > W$. The bounds pruning strategy uses a primal bound $\bar{c}$ that it is updated with the value of the best solution found so far. If $c(\mathcal{P}_{si}) + c(\mathcal{P}_{it}^{c}) \geq \bar{c}$, then path $\mathcal{P}_{si}$ can be safely pruned because a better (or equal) solution has been found before in the exploration. Finally, for dominance pruning, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two partial paths at a given node $v_i \in \mathcal{N}$. It is said that $\mathcal{P}_1$ dominates $\mathcal{P}_2$ if $c(\mathcal{P}_1) \leq c(\mathcal{P}_2)$ and $w(\mathcal{P}_1) < w(\mathcal{P}_2)$; or $c(\mathcal{P}_1) < c(\mathcal{P}_2)$. and $w(\mathcal{P}_1) \leq w(\mathcal{P}_2)$. For further information about these strategies, the reader is referred to Lozano & Medaglia (2013) and Smith et al. (2012).

*3.2 Path completion*

Given a partial path $P_{si}$ arriving to a node $v_i \in N$, the path completion strategy adds the minimum cost path $P_{it}^{c}$ from $v_i$ to $v_t$ to the partial path $\mathcal{P}_{si}$, i.e., $\mathcal{P}_{st} = \mathcal{P}_{si} \cup \mathcal{P}_{it}^{c}$ and checks if the completed path $\mathcal{P}_{st}$ is feasible and $c(\mathcal{P}_{st})$ less than the primal bound $\bar{c}$. Note that if a path completion occurs, there is no need to explore additional paths beginning with $\mathcal{P}_{si}$ because $\mathcal{P}_{it}^{c}$ is already the minimum cost path from $v_i$ to $v_t$ and thus $\mathcal{P}_{st}$ will be the minimum cost path beginning with partial path $\mathcal{P}_{si}$. In this case the pulse associated with $\mathcal{P}_{si}$ can be pruned and the primal bound can be updated. This idea was implemented in the pulse framework following the intuition suggested by Smith et al. (2012) as a future extension to their algorithm.

Furthermore, if a path cannot be completed using the minimum cost path, there is still a chance to update the primal bound. Given a partial path $\mathcal{P}_{si}$, the path can be completed by adding the minimum-weight feasible path $\mathcal{P}_{it}^{w}$ from $v_i$ to $v_t$ to the partial path, i.e., $\mathcal{P}_{st} = \mathcal{P}_{si} \cup \mathcal{P}_{it}^{w}$. If $c(\mathcal{P}_{st}) < \bar{c}$, the primal bound can be updated, but the pulse associated with path $\mathcal{P}_{si}$ cannot be pruned.

*3.2 Pulse queueing*

Breadth-first and depth-first search have both advantages and limitations (Korf, 1985). Breadth-first search (BFS) expands the start node through the outgoing arcs, and then processes the nodes by depth level, layer by layer, until it reaches the end node. The main drawbacks of BFS are: 1) the memory requirements to store all the states; and 2) the possible lack of feasible solutions at intermediate steps of the search (i.e., it may take until the last iteration to reach the end node). On the other hand, depth-first search (DFS) processes an outgoing arc from the most recently expanded node until it reaches the end node; then it backtracks to process unexplored outgoing arcs. In contrast to BFS, the only information stored in DFS relates to the currently explored path; thus it requires less memory requirements than BFS. Because DFS favors depth over breadth, it

reaches the end node often, thus it generates complete solutions that are used to update the primal bound. However, a major disadvantage of DFS is that it could waste time exploring unpromising regions of the search space before backtracking and correcting poor decision made at earlier stages of the exploration.

Because it is not obvious on which instances BFS or DFS give better results, we propose to combine both search paradigms. The idea is to perform a depth-first search from the start node but restricted with a maximum depth $\delta$. When a pulse reaches that depth, the partial path is stored in the pulse queue $\Sigma$ following a given queue discipline. Once there are no active pulses, we explore the queued pulses until $\Sigma$ is empty.

If the value for $\delta$ is large enough, the exploration behaves as DFS. Thus, if the first extended arc in a path $\mathcal{P}$ is not part of the optimal solution, it may be necessary to explore all the possible paths that include that arc to conclude that extending it was a naïve decision. However, by fixing a low value for $\delta$, the algorithm behaves much more like BFS and it is able to timely reroute the exploration after reaching the depth limit.

*3.3 Enhanced exploration order*

The graph exploration order is critical for the algorithm's performance and it is defined by the queue discipline for $\Sigma$. Given a partial path $\mathcal{P}_{si}$ arriving to node $v_i$, we considered the following queueing disciplines: 1) minimum cost, 2) maximum cost, 3) minimum weight, 4) maximum weight, and 5) best promise. The first four disciplines were presented by Smith et al. (2012) and the latter is our proposed exploration order. We define the promise of a path $\psi(P_{si})$ as the cumulative cost of a path $\mathcal{P}_{si}$ plus the cost of the minimum cost path from $v_i$ to $v_t$, formally $\psi(\mathcal{P}_{si}) = c(\mathcal{P}_{si}) + c(\mathcal{P}_{it}^c)$. Given that $\mathcal{P}_{it}^c$ is the best possible path from node $v_i$ to $v_t$, the rationale behind this queue discipline is to explore first those partial paths which promise the best possible objective function, that is, those paths with the minimum $\psi(\mathcal{P}_{si})$. It is remarkable that this exploration order tries to reach the end node as fast as possible and attain a tight primal bound opposed to the minimum cost or minimum weight criteria that explore the graph by layers.

Finally, it is important to note that each one of the acceleration strategies has an important effect on the algorithm's performance; however, its efficiency is not the sum of the individual effects of each strategy, but the interaction among all the acceleration strategies and the basic pruning strategies.

***4. Computational experiments***

We coded our pulse algorithm and the LC algorithm by Smith et al. (2012) to conduct a head-to-head comparison. Both algorithms were coded in Java, using Eclipse SDK version 3.6.1 and tested on a computer with an Intel Xeon X5450 @ 3.00 GHz (2 processors with four cores each) with 6 GB of RAM allocated to the memory heap size of the Java Virtual Machine on Windows Vista Professional.

We design two sets of computational experiments. The first set validates our implementation of the LC algorithm, showing that it is efficient and suitable as a benchmark for comparison. The second set measures the contributions of the acceleration strategies over a vast set of real-road networks.

*4.1 The testbed*

For the validation experiment we use randomly generated grid networks following the generation procedure proposed by Smith et al. (2012). We generated 30 instances for grids of 100x100, 200x200, and 400x400 nodes for a grand total of 90 instances. Each instance has a start node with outgoing arcs to the first layer of the grid and an end node with incoming arcs from the last layer.

For the second experiment we used real road networks, three from Raith & Ehrgott (2009) and 10 from the 9th DIMACS Implementation Challenge for the shortest path problem (Demetrescu et al., 2006). The networks were divided in two sets based on their size: medium and large. The medium-sized networks have between 9,000 and 1,000,000 nodes and the large-sized networks contain instances with over 1,000,000 nodes. Following the same approach by Smith et al. (2012), each network was adapted to the WCSPP-R by randomly selecting replenishment arcs with a probability of 5%. For each adapted network, we generated 30 instances with randomly selected start and end nodes for a grand total of 390 instances. Table 1 summarizes the size of the road networks used in this experiment.

**Table 1. Description of the benchmark with real road networks**

| Size | Road network | Description | Nodes | Arcs |
|---|---|---|---|---|
| Medium | DC | Washington DC | 9,559 | 39,377 |
| | RI | Rhode Island | 53,658 | 192,084 |
| | NY | New York | 264,346 | 733,846 |
| | BAY | San Francisco Bay Area | 321,270 | 800,172 |
| | NJ | New Jersey | 330,386 | 1,202,458 |
| | COL | Colorado | 435,666 | 1,057,066 |
| Large | FLA | Florida | 1,070,376 | 2,712,798 |
| | NW | Northwest USA | 1,207,945 | 2,840,208 |
| | NE | Northeast USA | 1,524,453 | 3,897,636 |
| | CAL | California and Nevada | 1,890,815 | 4,657,742 |
| | LKS | Great Lakes | 2,758,119 | 6,885,658 |
| | E | Eastern USA | 3,598,623 | 8,778,114 |
| | W | Western USA | 6,262,104 | 15,248,146 |

We tested different levels of tightness for the resource constraint. Smith et al. (2012) define the right side of the resource constraint as $W = \alpha W^- + (1-\alpha)W^+$ where $W^-$ is the lowest resource limit for which there is a feasible path, $W^+$ is the lowest resource limit for which the minimum cost path is the optimal solution, and $\alpha \in [0,1]$. We used four values for $\alpha$: 0.1, 0.5, 0.9, and 1; where small (large) values for $\alpha$ lead to loosely (tightly) constrained problems. Additionally, we characterized each instance by computing the exact values of $W^-$ and $W^+$. We have made publicly available all instances used in this paper for the WCSPP-R at http://hdl.handle.net/1992/1159.

*4.2 Validation experiment*

Smith et al. (2012) tested the performance of 10 different label treatment criteria and concluded that the minimum $c(\mathcal{P})$ is a strong alternative to solve the WCSPP-R. For this reason, we also chose this criterion in our Java implementation of the LC algorithm (jLC).

Table 2 compares the time (including preprocessing) reported by Smith et al. (2012) with their implementation of the LC algorithm (cLC) against our implementation in Java (jLC) over the set of randomly generated grids. For the sake of fairness, we scaled all our times using the LINPACK benchmark (Dongarra, 2009); according to this benchmark our computer is 1.54 times faster than the one used by Smith et al. (2012). Column 1 shows the name of the instance, column 2 presents the tightness of the resource constraint $\alpha$, column 3 shows the average time in seconds reported by Smith et al. (2012), and columns 4 and 5 report the scaled average time in seconds and the standard deviation for jLC over the 30 instances generated for each grid configuration.

**Table 2. Validation of our implementation against the original LC algorithm**

| Instance | $\alpha$ | cLC (s) | jLC (s) | |
|---|---|---|---|---|
| | | | avg. | s.d. |
| 100x100 | 0.1 | 0.21 | 0.02 | 0.00 |
| | 0.5 | 0.19 | 0.04 | 0.04 |
| | 0.9 | 0.13 | 0.10 | 0.01 |
| | 1 | 0.12 | 0.01 | 0.00 |
| 200x200 | 0.1 | 7.96 | 0.09 | 0.01 |
| | 0.5 | 7.85 | 0.37 | 0.35 |
| | 0.9 | 5.25 | 0.58 | 0.04 |
| | 1 | 4.67 | 0.07 | 0.01 |
| 400x400 | 0.1 | 10.41 | 0.39 | 0.04 |
| | 0.5 | 9.56 | 3.08 | 2.40 |
| | 0.9 | 8.62 | 3.85 | 0.47 |
| | 1 | 8.05 | 0.30 | 0.03 |

Table **2** shows that jLC is a strong and robust implementation of the algorithm. For all instances and values of $\alpha$ jLC outperforms cLC. Moreover, when the network size increases, the time gaps become even larger. This good performance is due to a thorough selection of the data structures and a careful implementation of the procedures used for managing the labels set.

*4.3 The pulse algorithm against the LC benchmark*

After validating our jLC implementation of the algorithm by Smith et al. (2012) we conducted a head-to-head comparison between jLC and our enhanced pulse algorithm. Given that the preprocessing procedure is the same for both algorithms, we decided to separate the preprocessing time from the rest of the execution time. For these experiments, jLC still uses the minimum $c(\mathcal{P})$ as label treatment criterion, while the pulse follows an exploration order by minimum promise $\psi(\mathcal{P})$. After fine tuning the pulse algorithm, the depth limit $\delta$ was fixed in 2. Table 3 and Table 4 present the results of these experiments. Column 1 shows the name of the

instance; column 2 presents the tightness of the resource constraint $\alpha$; column 3 presents the average time in seconds spent in the preprocessing procedure (same for jLC and for pulse); columns 4 and 5 present the average time in seconds used by jLC and pulse after the preprocessing procedure; column 6 presents the arithmetic mean of the speedup calculated as the ratio between the times for jLC and the pulse; columns 7 and 8 present the minimum and maximum speedup achieved among the corresponding 30 experiments; and column 9 presents the geometric mean of the speedup. Note that the geometric mean, opposed to the arithmetic mean, avoids the bias produced by the performance of few good results (Bixby, 2012). Finally, the last row presents an overall mean of the speedups over the whole testbed.

**Table 3. Assessing the effects of the acceleration strategies over the medium-size road networks**

| Instance | $\alpha$ | $Time_{pre}$ (s) | jLC (s) | Pulse (s) | Mean speedup | Min speedup | Max speedup | Geometric mean speedup |
|---|---|---|---|---|---|---|---|---|
| DC | 0.1 | 0.01 | <0.01 | <0.01 | 1.01 | 1.00 | 1.34 | 1.01 |
| | 0.5 | 0.01 | <0.01 | <0.01 | 2.58 | 0.65 | 22.81 | 1.51 |
| | 0.9 | 0.01 | <0.01 | <0.01 | 2.19 | 1.00 | 5.84 | 1.79 |
| | 1 | 0.01 | <0.01 | <0.01 | 1.02 | 0.34 | 2.01 | 0.96 |
| RI | 0.1 | 0.12 | <0.01 | <0.01 | 1.14 | 0.79 | 5.50 | 1.05 |
| | 0.5 | 0.12 | 0.02 | 0.01 | 4.54 | 0.74 | 40.34 | 1.99 |
| | 0.9 | 0.12 | 0.05 | 0.03 | 7.30 | 0.90 | 117.71 | 3.06 |
| | 1 | 0.11 | 0.05 | 0.04 | 2.35 | 0.74 | 15.04 | 1.62 |
| NY | 0.1 | 0.47 | <0.01 | <0.01 | 5.90 | 1.00 | 46.35 | 1.84 |
| | 0.5 | 0.47 | 0.05 | <0.01 | 15.10 | 1.00 | 63.72 | 6.91 |
| | 0.9 | 0.45 | 0.11 | 0.08 | 2.98 | 0.58 | 9.89 | 2.23 |
| | 1 | 0.43 | 0.11 | 0.10 | 1.46 | 0.74 | 5.80 | 1.28 |
| BAY | 0.1 | 0.54 | 0.02 | <0.01 | 15.18 | 0.84 | 182.45 | 2.29 |
| | 0.5 | 0.54 | 0.10 | 0.01 | 41.23 | 0.95 | 307.19 | 9.06 |
| | 0.9 | 0.50 | 0.11 | 0.09 | 3.66 | 0.51 | 17.74 | 2.03 |
| | 1 | 0.47 | 0.15 | 0.17 | 1.40 | 0.47 | 4.13 | 1.17 |
| NJ | 0.1 | 1.29 | <0.01 | <0.01 | 1.03 | 0.74 | 2.24 | 1.02 |
| | 0.5 | 1.16 | 0.20 | 0.03 | 58.20 | 0.94 | 1,459.07 | 3.41 |
| | 0.9 | 0.86 | 0.75 | 0.42 | 7.47 | 0.94 | 100.13 | 2.92 |
| | 1 | 0.76 | 0.50 | 0.34 | 2.27 | 0.85 | 14.78 | 1.62 |
| COL | 0.1 | 0.75 | 0.06 | <0.01 | 23.19 | 0.62 | 360.58 | 2.78 |
| | 0.5 | 0.75 | 0.15 | 0.02 | 34.36 | 0.61 | 299.69 | 8.42 |
| | 0.9 | 0.76 | 0.24 | 0.13 | 7.16 | 0.82 | 38.21 | 3.52 |
| | 1 | 0.84 | 0.36 | 0.31 | 3.42 | 0.71 | 26.15 | 1.82 |
| Global avg | | | | | 10.26 | | | 2.72 |

The results presented in Table 3 show that the pulse consistently outperforms jLC in 23 out of 24 midsized real-road network instances and through all values of the tightness factor $\alpha$. The acceleration strategies can achieve average speedups of up to 58 while the geometric average speedups range between 0.96 and 9, proving that these strategies really pay back on real-road networks. It is noticeable that the effects of the acceleration strategies present a high variability, for example, in the NJ network with $\alpha = 0.5$ the minimun speedup is 0.94 while the maximum speedup exceeds 1,400. However, both the arithmetic and geometric averages of speedup indicate a global speedup of roughly 10 and 3 times, respectively.

**Table 4. Computational results for the large size road networks**

| Instance | $\alpha$ | $Time_{pre}$ (s) | jLC (s) | Pulse (s) | Mean speedup | Min speedup | Max speedup | Geometric mean speedup |
|---|---|---|---|---|---|---|---|---|
| FLA | 0.1 | 2.28 | 0.23 | <0.01 | 36.22 | 0.71 | 559.22 | 2.64 |
| | 0.5 | 2.34 | 0.60 | 0.06 | 42.27 | 0.30 | 301.65 | 8.52 |
| | 0.9 | 2.30 | 1.05 | 0.73 | 3.43 | 0.38 | 25.40 | 2.02 |
| | 1 | 2.28 | 0.86 | 0.89 | 1.55 | 0.28 | 8.58 | 1.21 |
| NW | 0.1 | 2.46 | 0.14 | <0.01 | 38.42 | 0.65 | 967.13 | 1.57 |
| | 0.5 | 2.48 | 0.45 | 0.05 | 46.19 | 0.94 | 326.09 | 11.43 |
| | 0.9 | 2.41 | 0.65 | 0.29 | 5.79 | 0.72 | 38.26 | 2.87 |
| | 1 | 2.31 | 0.86 | 0.89 | 1.79 | 0.48 | 9.97 | 1.30 |
| NE | 0.1 | 3.62 | 0.12 | <0.01 | 33.21 | 0.86 | 623.38 | 2.81 |
| | 0.5 | 3.50 | 0.87 | 0.04 | 57.80 | 0.30 | 384.13 | 13.69 |
| | 0.9 | 3.54 | 1.09 | 0.91 | 4.41 | 0.63 | 49.67 | 1.97 |
| | 1 | 3.24 | 1.22 | 1.13 | 1.33 | 0.54 | 5.25 | 1.18 |
| CAL | 0.1 | 4.60 | 0.07 | <0.01 | 53.00 | 0.93 | 860.82 | 1.96 |
| | 0.5 | 4.80 | 1.38 | 0.04 | 127.59 | 0.32 | 824.32 | 18.78 |
| | 0.9 | 5.17 | 2.04 | 1.10 | 6.95 | 0.37 | 88.35 | 2.86 |
| | 1 | 5.44 | 3.76 | 1.39 | 2.84 | 0.56 | 26.03 | 1.73 |
| LKS | 0.1 | 6.71 | 0.47 | 0.05 | 27.83 | 0.67 | 410.84 | 2.16 |
| | 0.5 | 7.86 | 4.02 | 0.31 | 98.57 | 1.00 | 921.55 | 28.40 |
| | 0.9 | 11.04 | 3.35 | 3.26 | 4.38 | 0.15 | 34.19 | 1.87 |
| | 1 | 8.43 | 2.95 | 3.70 | 1.55 | 0.26 | 6.70 | 1.14 |
| E | 0.1 | 14.83 | 0.29 | <0.01 | 219.64 | 0.87 | 6,025.31 | 2.17 |
| | 0.5 | 13.35 | 2.32 | 0.16 | 55.12 | 0.43 | 295.87 | 20.73 |
| | 0.9 | 16.03 | 3.63 | 2.90 | 4.24 | 0.37 | 31.57 | 2.43 |
| | 1 | 12.73 | 2.69 | 4.42 | 1.26 | 0.28 | 3.13 | 1.04 |
| W | 0.1 | 43.38 | 0.82 | 0.04 | 145.46 | 0.23 | 2,831.51 | 4.53 |
| | 0.5 | 62.96 | 5.04 | 0.56 | 100.81 | 0.24 | 936.14 | 14.51 |
| | 0.9 | 64.25 | 8.59 | 2.38 | 6.69 | 0.75 | 48.55 | 3.74 |
| | 1 | 57.68 | 11.99 | 11.66 | 1.59 | 0.25 | 12.24 | 1.12 |
| Global avg | | | | | 40.36 | | | 5.73 |

Table 4 shows that the pulse also outperforms jLC in the 28 experiments made on the large sized instances and across all values of the tightness factor $\alpha$. Note that the average speedups in these networks are larger than those obtained in the midsized networks, thus showing a sign of scalability of the pulse algorithm. Additionally, despite the variability exhibited by the accelerated pulse with minimum and maximum speedups between 0.25 and 6,000, it is evident the strength of the algorithm. In fact, the arithmetic and geometric average speedups are up to 219 and 28. Moreover, the pulse reaches a global speedup of roughly 40 and 6 measured by the arithmetic and geometric mean respectively. Finally, it is remarkable the time spent solving most of the instances is about 15 seconds, and around 1 minute for the largest instance with up to 6 million nodes and 15 million arcs.

## *5. Concluding remarks*

We present a set of acceleration strategies for the WSPP-R extending the work by Smith et al. (2012) and Lozano & Medaglia (2013). First, we provide a *path completion strategy* that avoids unnecessary exploration of suboptimal regions of the solution space (paths in the network) and strengthens the primal bound. Second, we combine the ideas of performing breadth- and depth-first search via a pulse queue that controls the depth of the exploration. Finally, we propose an enhanced graph exploration order based on a promise obtained from dual bounds.

From a computational perspective, we adapted and characterized a vast set of real-road networks that now comprises a comprehensive publicly available testbed for the WCSPP-R. We conducted several computational experiments that demonstrate the effects of the proposed enhanced pulse algorithm with acceleration strategies compared against the state-of-the-art algorithm for the WCSPP-R. In the midsized networks the acceleration strategies reached average speedups of up to 58 and they exhibit an overall performance of 10 and 3 times faster measured by the arithmetic and the geometric mean of speedups, respectively. In the larger networks the effects of the acceleration strategies are even better, achieving global speedups of roughly 40 and 6 measured by the arithmetic and geometric mean, respectively; moreover, they achieved average speedups of up to 219 times on networks with up to 6 million nodes and 15 million arcs. Finally, our acceleration strategies are easy to understand and to implement and could be applied both on the pulse algorithm or other label-based algorithms like LC.

## References


*Azadeh, A., Farahani, M. H., Eivazy, H., Nazari-Shirkouhi, S., & Asadipour, G. (2013). A hybrid meta-heuristic algorithm for optimization of crew scheduling . Applied Soft Computing , 13(1), 158-164.*

*Bixby, R. E. (2002). Solving real-world linear programs: A decade and more of progress. Operations research, 50(1), 3-15.*

*Cabral, E. A. (2005). Wide area telecommunication network design: problems and solution algorithms with application to the Alberta SuperNet. Ph.D. dissertation, University of Alberta, Canada.*

*Cabral, E. A., Erkut, E., Laporte, G., & Patterson, R. A. (2008). Wide area telecommunication network design: application to the Alberta SuperNet. Journal of the Operational Research Society, 59, 1460-1470.*

*Corneil, D. (2005). Lexicographic Breadth First Search: A Survey. In Graph-Theoretic Concepts in Computer Science (Vol. 3353, pp. 1-19). Springer Berlin Heidelberg.*

*Demetrescu, C., Goldberg, A., & Johnson, D. (2006). 9th DIMACS Implementation Challenge - Shortest Paths. www.dis.uniroma1.it/~challenge9/*

*Dongarra, J. (2009). Performance of various computers using standard linear equations software. Tech. rep., University of Tennessee, USA.*

*Dumitrescu, I., & Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem. Networks, 42(3), 135-153.*

*Eisner, J., Funke, S., & Storandt, S. (2011). Optimal route planning for electric vehicles in large networks. Proceedings of the Twenty-Fifth AAAI conference on Artificial Intelligence.*

*Gallardo-Lozano, J., Milans-Montero, M. I., Guerrero-MartÃnez, M. A., & Romero-Cadaval, E. (2012). Electric vehicle battery charger for smart grids. Electric Power Systems Research, 90, 18-29.*

*Handler, G. Y., & Zang, I. (1980). A dual algorithm for the constrained shortest path problem. Networks, 10(4), 293-309.*

*Joksch, H. C. (1966). The shortest route problem with constraints(Shortest route problem with constraint, using set of nodes). Journal of Mathematical analysis and applications, 14, 191-197.*

*Kobayashi, Y., Kiyama, N., Aoshima, H., & Kashiyama, M. (2011). A route search method for electric vehicles in consideration of range and locations of charging stations. Intelligent Vehicles Symposium (IV), 2011 IEEE, (pp. 920-925).*

*Konak, A. (2012). Network design problem with relays: A genetic algorithm with a path-based crossover and a set covering formulation . European Journal of Operational Research , 218(3), 829-837.*

*Korf, R. E. (1985). Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence, 27, 97-109.*

*Laporte, G., & Pascoal, M. M. (2011). Minimum cost path problems with relays. Computers & Operations Research, 38(1), 165-173.*

*Lee, J., & Park, G.-L. (2013). Orienteering Problem Modeling for Electric Vehicle-Based Tour. In Intelligent Information and Database Systems (Vol. 7803, pp. 100-108). Springer Berlin Heidelberg.*

*Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. Computers & Operations Research, 40(1), 378-384.*

*Raith, A., & Ehrgott, M. (2009). A comparison of solution strategies for biobjective shortest path problems . Computers & Operations Research , 36(4), 1299-1331.*

*Santos, L., Coutinho-Rodrigues, J., & Current, J. R. (2007). An improved solution algorithm for the constrained shortest path problem . Transportation Research Part B: Methodological , 41(7), 756-771.*

*Smith, O. J., Boland, N., & Waterer, H. (2012). Solving shortest path problems with a weight constraint and replenishment arcs. Computers & Operations Research, 39(5), 964-984.*

*Wang, Y., & Pham, H. (2013). Maintenance Modeling and Policies. Stochastic Reliability and Maintenance Modeling (Vol. 9, pp. 141-158). Springer London*