A branch-and-price algorithm for the Minimum Latency Problem

Teobaldo Bulhões, Ruslan Sadykov, Eduardo Uchoa

Volume 2017, Number 4

July, 2017
A branch-and-price algorithm for the Minimum Latency Problem

Teobaldo Bulhões\textsuperscript{a}, Ruslan Sadykov\textsuperscript{b}, Eduardo Uchoa\textsuperscript{c}\textsuperscript{*}

\textsuperscript{a}Instituto de Computação, Universidade Federal Fluminense, Brazil
\textsuperscript{b}Inria Bordeaux - Sud Ouest, France
\textsuperscript{c}Departamento de Engenharia de Produção, Universidade Federal Fluminense, Brazil

Abstract

This paper deals with the Minimum Latency Problem (MLP), a variant of the well-known Traveling Salesman Problem in which the objective is to minimize the sum of waiting times of customers. This problem arises in many applications where customer satisfaction is more important than the total time spent by the server. This paper presents a novel branch-and-price algorithm for MLP that strongly relies on new features for the $ng$-path relaxation, namely: (1) a new labeling algorithm with an enhanced dominance rule named multiple partial label dominance; (2) a generalized definition of $ng$-sets in terms of arcs, instead of nodes; and (3) a strategy for decreasing $ng$-set size when those sets are being dynamically chosen. Also, other elements of efficient exact algorithms for vehicle routing problems are incorporated into our method, such as reduced cost fixing, dual stabilization, route enumeration and strong branching. Computational experiments over TSPLIB instances are reported, showing that several instances not solved by the current state-of-the-art method can now be solved.

Keywords: minimum latency, $ng$-paths, branch-and-price

1. Introduction

This paper deals with the Minimum Latency Problem (MLP). In MLP, we are given a complete directed graph $G = (V, A)$ and a time $t_{ij}$ for each arc $(i, j) \in A$. Set $V$ is composed of $n + 1$ nodes: node 0, representing a depot, and $n$ nodes representing customers. The task is to find a Hamiltonian circuit $(i_0 = 0, i_1, \ldots, i_n, i_{n+1} = 0)$, a.k.a tour, in $G$ that minimizes $\sum_{t=1}^{n+1} l(i_t)$, where the latency $l(i_t)$ is defined as the accumulated travel time from the depot to $i_t$. The MLP is related to the Time Dependent Traveling Salesman Problem (TDTSP), a generalization of the Traveling Salesman Problem (TSP) in which the cost for traversing an arc depends on its position in the tour. More precisely, MLP can be viewed as the particular case of the TDTSP where the cost of an arc $(i, j)$ in position $p, 0 \leq p \leq n$, is given by $(n - p + 1)t_{ij}$.

\textsuperscript{*}Corresponding author

Email addresses: tbulhoes@ic.uff.br (Teobaldo Bulhões), Ruslan.Sadykov@inria.fr (Ruslan Sadykov), uchoa@producao.uff.br (Eduardo Uchoa)

Research papers in Cadernos do LOGIS-UFF are not peer reviewed, authors are responsible for their contents.
The MLP is also known in the literature as Delivery Man Problem (Roberti and Mingozzi, 2014), Traveling Repairman Problem (Afrati, Foto et al., 1986), Traveling Deliveryman Problem (Tsitsiklis, 1992) and Traveling Salesman with Cumulative Costs (Bianco et al., 1993). Although MLP seems to be a simple variant of TSP, some important characteristics are very different in those problems. First, two different viewpoints of a distribution system are considered: TSP is server oriented, since one wants to minimize the total travel time; on the other hand, MLP is customer oriented because the objective is equivalent to minimizing the average waiting time of customers (Silva et al., 2012; Sitters, 2002; Archer and Williamson, 2003). Customer satisfaction is the main objective in many applications, such as home delivery services (Méndez-Díaz et al., 2008), and has attracted the attention of researchers, as reflected by the considerable number of MLP variants studied in the very last years (see, for instance, (Lysgaard and Wohlk, 2014; Rivera et al., 2016; Nucamendi-Guillén et al., 2016; Sze et al., 2017)). Second, in contrast to what happens in TSP, simple local changes may affect globally a MLP solution because the latency of subsequent customers may change (Silva et al., 2012; Sitters, 2002). This can make it more difficult to solve MLP both exactly and heuristically. For example, current state-of-the-art exact methods for MLP are not capable of solving consistently instances with 150 customers, whereas TSP instances with thousands of customers are solved routinely (Abeledo et al., 2013).

Many complexity results for MLP have been obtained. The problem is NP-Hard for general metric spaces (Sahni and Gonzalez, 1976), and remains NP-Hard even if the times correspond to euclidean distances (Afrati, Foto et al., 1986) or if they are obtained from an underlying graph that is a tree (Sitters, 2002). On the other hand, the problem is polynomial if the underlying graph is a path (Afrati, Foto et al., 1986; Garcia et al., 2002), a tree with equal weights or a tree with diameter at most 3 (Blum et al., 1994). The MLP with deadlines, i.e., with upper bounds on \( t(i) \), is NP-Hard even for paths (Afrati, Foto et al., 1986). In terms of approximation, hardness results show that one should not expect to attain arbitrarily good approximation factors for MLP (Blum et al., 1994). However, 3.59 and 3.03 approximations are known for general metric spaces and general trees, respectively (Chaudhuri et al., 2003; Archer and Blasiak, 2010). Moreover, a constant factor approximation is not likely to exist if times do not satisfy the triangle inequality, just as for TSP (Blum et al., 1994).

The first integer programming formulations were given in (Picard and Queyranne, 1978), where the authors stated TDTSP as a machine scheduling problem and solved instances with up to 20 jobs by means of a branch-and-bound method over lagrangian bounds. A new formulation with \( n \) constraints was presented in (Fox et al., 1980), but the authors did not report any computational results. Lucena (1990) and Bianco et al. (1993) followed the same approach as Picard and Queyranne (1978) and employed langragian bounds in experiments over MLP instances with up to 30 and 60 vertices, respectively. The latter authors also developed a dynamic programming method capable of attesting that the bounds obtained for 60-vertex instances were within 3% from optimality. Then, a series of enumerative strategies based on new formulations
was introduced in (Fischetti et al., 1993; Van Eijl, 1995; Méndez-Díaz et al., 2008; Bigras et al., 2008; Godinho et al., 2014), as well as cutting planes (Van Eijl, 1995; Méndez-Díaz et al., 2008; Bigras et al., 2008) and polyhedral studies (Méndez-Díaz et al., 2008). Instances with 60 vertices could already be solved by the algorithm of Fischetti et al. (1993). More recently, Abeledo et al. (2013) managed to solve almost all TSPLIB instances with up to 107 vertices using a branch-cut-and-price algorithm. The authors departed from a formulation by Picard and Queyranne (1978) and proposed new inequalities, that are proved to be facet-inducing. Roberti and Mingozzi (2014) implemented dual ascent and column generation techniques to compute a sequence of lower bounds associated with set partitioning formulations where a column represents a ng-path, which is a path relaxation introduced by Baldacci et al. (2011). A ng-path may contain cycles, but just those allowed by the so-called ng-sets. Such sets are iteratively augmented so that less cycles are allowed and improved bounds are obtained. The final lower bound is used in a dynamic programming recursion to compute the optimal solution. This method could solve some larger TSPLIB instances, with up to 150 vertices, and currently holds the status of state-of-the-art exact method for MLP.

Finally, heuristic algorithms for MLP can be found in (Ngueveu et al., 2010; Salehipour et al., 2011; Silva et al., 2012; Mladenović et al., 2013).

This paper presents a novel branch-and-price algorithm for MLP that strongly relies on ng-paths. Following the directions of Roberti and Mingozzi (2014), our method works over a set partitioning formulation where columns represent ng-paths and the column generation bounds computed on each node of the tree are derived from dynamically defined ng-sets. However, we introduce the following improvements on the use of ng-paths.

- **Multiple Partial Label Dominance:** In the labeling algorithms used for pricing ng-paths, a partial path \( P \) is represented as a label \( L(P) \). A key concept in this kind of algorithm is of dominance. A label \( L(P_1) \) dominates a label \( L(P_2) \) if the cost of \( P_1 \) is not larger than the cost of \( P_2 \) and every completion of \( P_2 \) is also a feasible completion of \( P_1 \). In this case, \( L(P_2) \) can be safely eliminated. In this paper, we propose a stronger dominance rule by which some extensions for \( L(P_2) \) can be avoided, even though this label can not be completely disregarded according to the classical dominance rule. We briefly discuss two alternative implementations of this new dominance rule, where the best one typically speeds the labeling algorithm by factors between 4 and 8.

- **Arc-Based ng-Path Relaxation:** ng-sets as originally defined by Baldacci et al. (2011) are a vertex-based memory mechanism. In this paper, we provide a generalized definition of them in terms of arcs. We show that this new definition is particularly useful in the context of dynamically defined ng-sets, allowing strong bounds to be obtained in more controlled pricing times.

- **Fully Dynamic ng-Path Relaxation:** We improve the dynamic ng-path relaxation of Roberti and Mingozzi (2014) by introducing a procedure for decreasing the ng-sets, without
changing the current bounds. Such reductions are beneficial for the pricing time and also help to refine the choice of \(ng\)-sets.

Also, other well-known elements of efficient exact algorithms for many other variants of the vehicle routing problem (VRP) are incorporated into our method, namely reduced cost fixing, dual stabilization, route enumeration and strong branching. Computational experiments over MLP instances derived from TSPLIB were conducted to attest the effectiveness of the new branch-and-price algorithm. The results show that better bounds can be obtained in less computational time when compared to the state-of-the-art algorithm, specially because of the new features for the \(ng\)-path relaxation. In particular, the branch-and-price solved all the 9 instances with up to 150 vertices not solved in Roberti and Mingozzi (2014). It could also solve 4 additional instances, with more than 150 vertices, never considered before by exact methods.

The remainder of this paper is organized as follows. Section 2 discusses the \(ng\)-path relaxation and labeling algorithms. Section 3 introduces the new features for the \(ng\)-path relaxation. The proposed branch-and-price algorithm is described in Section 4, where we also give implementations details. Computational experiments are presented in Section 5. Finally, concluding remarks are drawn in the last section.

2. Route Relaxations and Labeling Algorithms

This section reviews the route relaxations and labeling algorithms that are related to current state-of-the-art exact algorithms for VRPs, such as Capacitated VRP (CVRP), VRP with time windows (VRPTW), and the MLP itself. Such algorithms are based on a combination of column and cut generation over the following set-partitioning formulation.

\[
\begin{align*}
\min & \quad \sum_{r \in \Omega} c_r \lambda_r \\
\text{s.t.} & \quad \sum_{r \in \Omega} a_{i}^{r} \lambda_r = 1, & \forall i \in C, \\
& \quad \lambda_r \in \{0, 1\}, & \forall r \in \Omega,
\end{align*}
\]

where \(C\), \(\Omega\), \(c_r\) and \(a_{i}^{r}\) denote, respectively, the set of customers, the set of feasible routes, the cost of route \(r\), and the number of times route \(r\) visits customer \(i\).

As the number of variables in Formulation (1)-(3) is exponential on \(|C|\), column generation is typically applied to solve its linear relaxation. The pricing subproblem depends on the considered variant, but it can often be modeled as the Elementary Resource Constrained Shortest Path Problem (ERCSPP). In ERCSPP, we are given a directed graph \(G' = (V', A')\) with vertex set \(V'\) and arc set \(A'\); source and sink nodes \(s \in V'\) and \(t \in V'\), respectively; and a set of resources \(\mathcal{R}\). Each arc \((i, j)\) has an associated cost \(c_{ij} \in \mathbb{R}\) and consumes a predefined amount \(w_{ij}^{r} \in \mathbb{R}_{>0}\) of resource \(r\), for each \(r \in \mathcal{R}\). Moreover, for each \(i \in V'\) and \(r \in \mathcal{R}\), let \(l_{i}^{r}\) and \(u_{i}^{r}\) be, respectively,
the minimum and the maximum consumption of resource \( r \) in any partial path from \( s \) to \( i \). The
task is to find the least-cost path from \( s \) to \( t \) satisfying the resource constraints and containing
no cycles. Note that, in order to satisfy minimum consumption, it is possible to “drop resources”
at no cost. See (Di Puglia Pugliese and Guerriero, 2012) for further details on ERCSPP.

From now on, for ease of presentation, we assume the case related to MLP, where a single
discrete resource is present. The consumption of this single resource by an arc \( (i, j) \) will be
denoted as \( w_{ij} \in \mathbb{Z}_{\geq 0} \) and the lower and upper bounds on its consumption when reaching vertex \( i \)
as \( l_i \in \mathbb{Z}_{\geq 0} \) and \( u_i \in \mathbb{Z}_{\geq 0} \). In the graph \( G^e \) for MLP, nodes in \( \{1, \ldots, n\} \) represent customers, while
source and sink nodes are associated with the depot. The single resource indicates the number
of arcs in a partial path, and hence \( w_{ij} = 1 \) for any arc \( (i, j) \in A^e \). Finally, \((l_i, u_i) = (1, n)\) for a
vertex \( i \in \{1, \ldots, n\} \); \((l_s, u_s) = (0, 0)\) and \((l_t, u_t) = (n + 1, n + 1)\).

ERCSPP is NP-Hard in the strong sense (Dror, 1994) and also a difficult problem to solve
in practice. The “offending” constraint is the one that imposes elementarity, since the Resource
Constrained Shortest Path Problem (RCSPP) can be solved in pseudo-polynomial time. RCSPP
is a relaxation of ERCSPP where a vertex can be visited more than once in an optimal path, as
long as the resource constraints are satisfied. In view of this, several state-of-the-art algorithms
employ some route relaxation as an alternative to elementary routes. The idea is to replace set \( \Omega \)
by some set also containing non-elementary routes so as to make the pricing subproblem easier. An
ideal relaxation would provide the elementary route bound while keeping the pricing subproblem
tractable. It is worth mentioning that such a relaxation is kind of mandatory in problems where
the optimal solution has exactly one route (e.g., MLP), otherwise the pricing subproblem is as
hard as the original problem, rendering column generation meaningless.

The first route relaxation is just to allow any non-elementary route, as long as it satisfies the
resource constraints. As the elementarity is relaxed, the pricing corresponds to a RCSPP, which
can be solved in pseudo-polynomial time. It was already observed in Christofides et al. (1981)
that it is possible to eliminate routes with 2-cycles (subpaths like \( i \to j \to i \)) without increasing
the complexity. The bounds obtained with 2-cycle elimination may be good in some cases, but
are likely to be poor in other cases, specially in routing problems with many customers per route
(Martinelli et al., 2014). This motivated Irnich and Villeneuve (2006) to propose an algorithm
to forbid cycles of an arbitrary maximum size \( k \). The pricing subproblem now is referred to as
RCSPP with \( k \)-cycle elimination. Theoretically, the proposed algorithm can be used for pricing
elementary routes, but only small values of \( k \) can be efficiently used in practice. This is because the
complexity of the algorithm grows by a factor of up to \( k^2.k! \). Nevertheless, \( k \)-cycle elimination for
small values of \( k \) proved to be useful at that time, improving column generation based algorithms
for several VRP variants. For example, the branch-cut-and-price for MLP in (Abeledo et al.,
2013) uses \( k = 5 \).

Later, Baldacci et al. (2011) introduced a new kind of elementarity relaxation, the so-called
\textit{ng}-routes. Extensive experiments on several VRP variants show that \textit{ng}-routes are almost al-
ways more efficient than routes without \textit{k}-cycles, in the sense of providing better bounds in less computational time. In contrast to previous relaxations, cycles eliminated by \textit{ng}-routes are not distinguished by size. Instead, a cycle \( H = (i_0, i_1, \ldots, i_p = i_0) \) is forbidden if all vertices \( i_1, \ldots, i_{p-1} \) are able to “remember” vertex \( i_0 \). Formally speaking, we define an \textit{ng}-set \( N_i \) for each vertex \( i \). We assume that \( i \in N_i \). Typically, \( N_i \) contains the vertices that are likely to appear close to vertex \( i \) in low-cost paths, e.g., nearest customers in VRPs to take advantage of a locality principle that is often present in those problems. In case \( j \in N_i \), we say that \( i \) remembers \( j \) or equivalently that \( j \) is remembered by \( i \). Then, each path \( P = (s = i_0, i_1, \ldots, i_p) \) has an associated set of forbidden extensions \( \Pi(P) \) that is computed based on the \textit{ng}-sets:

\[
\Pi(P) = \left\{ i_k \in C(P) \setminus \{i_p\} : i_k \in \bigcap_{s=k+1}^{p} N_i \right\} \cup \{i_p\} \tag{4}
\]

where \( C(P) \) is the set of vertices visited by \( P \). In words, a vertex \( i_k \neq i_p \) belongs to \( \Pi(P) \) if it is remembered by all vertices \( i_s \) with \( k < s \leq p \). Therefore, path \( P \) is \textit{ng}-feasible if \( i_k \notin \Pi(P_{k-1} = (i_0, i_1, \ldots, i_{k-1})), 1 \leq k \leq p \). Alternatively, one can define the \textit{ng}-path relaxation in terms of the \textit{ng}-memory \( M_i \) of a vertex \( i \), which is the set of vertices that remember \( i \), i.e., \( M_i = \{ j \in V : i \in N_j \} \). In this case, the set of forbidden extensions for path \( P \) is computed as in Equation (5). Moreover, path \( P \) is \textit{ng}-feasible if between two visits to a vertex \( i \), some vertex \( j \notin M_i \) is visited. In this paper, both concepts (\textit{ng}-sets and \textit{ng}-memories) are used. Clearly, the greater the \textit{ng}-memories, the closer to elementary are the \textit{ng}-paths.

\[
\Pi(P) = \left\{ i_k \in C(P) \setminus \{i_p\} : i_s \in M_{i_k}, s = k+1, \ldots, p \right\} \cup \{i_p\} \tag{5}
\]

The RCSPP with \textit{ng}-routes is commonly solved by means of a labeling algorithm. In this kind of algorithm, a label \( L(P) = (c(P), w(P), v(P), \Pi(P)) \) represents a partial path \( P \) ending at vertex \( v(P) \) with cost \( c(P) \), resource consumption \( w(P) \) (we are assuming a single-resource), and set \( \Pi(P) \) of forbidden extensions. A single label representing the path defined only by the source vertex \( s \) is present in the beginning, and new labels are generated by extending existing ones along all possible arcs. In order to the algorithm to be correct, it is sufficient to process existing labels in increasing order of resource consumption. Dominance rules are applied to eliminate labels not leading to optimal paths, where a label \( L(P_1) \) dominates a label \( L(P_2) \) if the cost of \( P_1 \) is less than or equal to the cost of \( P_2 \) and any feasible completion to \( P_2 \) into a feasible \( s - t \) path is also feasible to \( P_1 \). The general scheme of labeling algorithms with \textit{ng}-paths is shown in Algorithm 1, where \( L_j \) and \( U_j \) denote, respectively, the set of processed and unprocessed labels that correspond to paths ending at vertex \( j \).

The feasibility test in Line 5 of Algorithm 1 takes into account the resource constraints and the extensions forbidden by \textit{ng}-sets. More precisely, the new path \( P + k \) is feasible if \( l_k \leq w(P) + w_{ik} \leq u_k \) and \( k \notin \Pi(P) \). Regarding the dominance checks performed in lines 7 and 10, the following
Algorithm 1 General Labeling Algorithm with $ng$-paths

1: Initialize $L_j$ and $U_j$ as empty sets, for all $j \in V$
2: Add the initial label to $U_s$
3: while $\bigcup_{j \in V} U_j \neq \emptyset$ do
4: Choose a label $L(P) \in \bigcup_{j \in V} U_j$ with minimum resource consumption and let $i = v(P)$
5: for each $(i, k) \in A$ such that $P + k$ is feasible do
6: Define label $L(P + k) = (c(P) + c_{ik}, w(P) + w_{ik}, k, (\Pi(P) \cap N_k) \cup \{k\})$
7: if $L(P + k)$ is dominated by any label in $L_k \cup U_k$ then
8: continue
9: else
10: Remove labels in $U_k$ dominated by $L(P + k)$
11: $U_k \leftarrow U_k \cup \{L(P + k)\}$
12: $U_i \leftarrow U_i \setminus \{L(P)\}$
13: $L_i \leftarrow L_i \cup \{L(P)\}$
14: return best label in $L_t$

conditions are sufficient and necessary to verify that a label $L(P')$ dominates a label $L(P)$.

(I) $v(P') = v(P)$
(II) $c(P') \leq c(P)$
(III) $w(P') \leq w(P)$
(IV) $\Pi(P') \subseteq \Pi(P)$

3. New Features for the $ng$-Path Relaxation

This section presents new contributions on the use of the $ng$-path relaxation. Such contributions are not built over any strong assumption regarding MLP and can be used in several other problems.

3.1. Multiple Partial Label Dominance

We will now introduce a stronger dominance rule called multiple partial label dominance (MPLD). Suppose that condition (IV) of the classical dominance rule discussed in Section 2 is the only one not satisfied by $P$ and $P'$. Therefore, label $L(P)$ is not dominated by label $L(P')$ because there exists a vertex $i$ such that $P + i$ is $ng$-feasible, whereas $P' + i$ is not. While completely disregarding label $L(P)$ because of label $L(P')$ is not correct, some extensions for the former may be unnecessary. That is, label $L(P)$ may be partially dominated by label $L(P')$.

Proposition 1. Let $L(P)$ and $L(P')$ be two labels such that

- $v(P') = v(P) = i$
- $c(P') \leq c(P)$
• \( w(P') \leq w(P) \)
• \( \Pi(P') \not\subseteq \Pi(P) \)

Then any extension to a vertex \( j \notin \bigcup_{k \in \Pi'} M_k \) can be safely disregarded for \( L(P) \), where \( \Pi' = \Pi(P') \setminus \Pi(P) \).

**Proof.** Let \( j \) be a vertex such that \((i, j) \in A\) and \( j \notin \bigcup_{k \in \Pi'} M_k \). Since we have assumed that \( l \in M_l \) for every \( l \in V \), we have that \( j \notin \Pi' \). Hence, if \( P + j \) is \( ng \)-feasible, so is \( P' + j \) and clearly \( v(P' + j) = v(P + j) \), \( c(P' + j) \leq c(P + j) \) and \( w(P' + j) \leq w(P + j) \). Furthermore, as \( j \notin \bigcup_{k \in \Pi'} M_k \), any vertex \( k \in \Pi' \) is forgotten by \( P' + j \) and \( P + j \), and thus \( \Pi(P' + j) \subseteq \Pi(P + j) \). Therefore, label \( L(P' + j) \) dominates label \( L(P + j) \). \( \square \)

In view of Proposition 1, a label \( L(P) \) should also store a set \( \xi(P) \) representing the extensions that can be avoided because of partial dominance, which we call *dominated extensions*. Let \( \mathcal{L}(P) \) be the set of all labels \( L(P') \) that together with label \( L(P) \) satisfy conditions (I), (II) and (III). By applying the partial dominance from the multiple labels in \( \mathcal{L}(P) \), the resulting set of dominated extensions for label \( L(P) \) is:

\[
\xi(P) = \bigcup_{L(P') \in \mathcal{L}(P)} \left\{ j \in V : j \notin \bigcup_{k \in \Pi(P')} M_k \right\}
\]  

(6)

A small example of MPLD is presented in Figure 1. In this example, we have six nodes (besides source and sink nodes) and all labels ending at vertex 1 are depicted in the figure. Sets of dominated extensions for these labels are defined as \( \xi(P_1) = \emptyset \), \( \xi(P_2) = V \setminus M_2 \), \( \xi(P_3) = (V \setminus M_5) \cup (V \setminus (M_2 \cup M_3)) \) and \( \xi(P_4) = (V \setminus M_5) \cup (V \setminus M_3) \). All extensions for label \( L(P_4) \) are either forbidden because of \( ng \)-sets or dominated because of labels \( L(P_1) \), \( L(P_2) \) and \( L(P_3) \), therefore \( L(P_4) \) can be removed. This illustrates a situation where MPLD results in complete dominance.

\[
\begin{align*}
M_1 &= \{1, 2, 3, 4, 5, 6\} \\
M_2 &= \{1, 2, 5\} \\
M_3 &= \{1, 3, 4\} \\
M_4 &= \{1, 2, 3, 4\} \\
M_5 &= \{1, 2, 5, 6\} \\
M_6 &= \{1, 6\}
\end{align*}
\]

![](image1.png)

**Figure 1:** An example of MPLD. Left: \( ng \)-memories. Right: labels representing paths ending at vertex 1, all of them with the same resource consumption.
3.2. Arc-Based \textit{ng}-Path Relaxation

The main idea exploited by the \textit{ng}-path relaxation is that, in many problems, cycles are often confined to small “neighborhoods” of the graph: once a new visit to a node $i$ is performed by a partial path $P$, any node $j$ that is not a neighbor of $i$ is forgotten. The set of nodes remembered by $P$ is computed as a function of \textit{ng}-memories defined over the set of vertices $V$. In this section, we show a generalized definition of the \textit{ng}-path relaxation where \textit{ng}-memories are defined in terms of arcs instead of nodes. The inspiration for this new definition comes from the arc-based limited memory technique developed by Pecin et al. (2017) in order to reduce the impact of the non-robust Rank-1 Chvátal-Gomory cuts on the labeling algorithm.

We denote the arc-based \textit{ng}-memory of vertex $j$ as $\rightarrow \mathcal{M}_j$, which is the set of arcs that remember vertex $j$. Let $P = (a_0, a_1, \ldots, a_p)$ be a partial path composed of arcs $a_0 = (s = i_0, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1})$. Similarly to Equation (5), we now define the set of forbidden extensions for path $P$ as:

$$\overline{\Pi}(P) = \left\{ i_k \in C(P) \setminus \{i_{p+1}\} : a_s \in \rightarrow \mathcal{M}_{i_k}, s = k, \ldots, p \right\} \cup \{i_{p+1}\} \quad (7)$$

Sets $\Pi(P)$ and $\overline{\Pi}(P)$ are equivalent if one defines $\rightarrow M_k = \{(i, j) \in A : i \in M_k \land j \in M_k\}$ for every vertex $k \in V$. This generalized definition is particularly useful in the context of dynamically defined \textit{ng}-memories. In this setting, one wants to augment current \textit{ng}-memories in order to forbid a given cycle $H = (a_0 = (i_0, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1} = i_0))$. For doing so, vertices $i_1, \ldots, i_q$ should be added to the vertex-based \textit{ng}-memory $\rightarrow M_{i_0}$. However, this forbids not only $H$, but any cycle $H' = (i_0, \ldots, i_q)$ passing through a subset of $\{i_0, i_1, \ldots, i_p\}$, which may represent a considerable impact on the labeling algorithm. On the other hand, the impact of adding $a_0, \ldots, a_{p-1}$ to $\rightarrow M_{i_0}$ is much less considerable since only cycles $H' = (i_0, \ldots, i_q, i_0)$, with $q = 1, \ldots, p$, are forbidden — notice that it is not necessary to add $(i_q, i_0)$ to $\rightarrow M_{i_0}$ to forbid a cycle $H' = (i_0, \ldots, i_q, i_0)$ because $i_0 \in \overline{\Pi}((i_0, i_1, \ldots, i_q))$ if $a_0, \ldots, a_{q-1} \in \rightarrow M_{i_0}$. We will show in our computational experiments that this reduced impact is crucial for solving hard MLP instances.

Hereafter, the term \textit{ng}-memory refers to arc-based \textit{ng}-memory and we will explicitly indicate if we refer to vertex-based \textit{ng}-memory.

3.3. Fully Dynamic \textit{ng}-Path Relaxation

Let us consider (arc-based) \textit{ng}-memories $\rightarrow \mathcal{M}$ and define $\Omega(\rightarrow \mathcal{M})$ as the set of all feasible \textit{ng}-routes w.r.t $\rightarrow \mathcal{M}$. Notice that $\Omega \subseteq \Omega(\rightarrow \mathcal{M})$. As we have discussed before, the following relaxation of formulation (1)-(3), hereafter denoted $LP(\rightarrow \mathcal{M})$, is the basis of several state-of-the-art column generation based algorithms for vehicle routing problems.

$$LB(\rightarrow \mathcal{M}) = \min \sum_{r \in \Omega(\rightarrow \mathcal{M})} c_r \lambda_r \quad (8)$$
\[
\begin{align*}
\text{s.t.} \quad \sum_{r \in \Omega(\vec{\mathcal{M}})} a^i_r \lambda_r &= 1, & \forall i \in \mathcal{C}, & (9) \\
\lambda_r &\geq 0, & \forall r \in \Omega(\vec{\mathcal{M}}). & (10)
\end{align*}
\]

The quality of the bound LB(\vec{\mathcal{M}}) depends on the ng-memories \( \vec{\mathcal{M}} \). Ideally, one should define \( \vec{\mathcal{M}} \) so as to guarantee that LB(\vec{\mathcal{M}}) corresponds to the bound attained if \( \Omega(\vec{\mathcal{M}}) \) is replaced by \( \Omega \) in the formulation, i.e., if only elementary routes are generated in the pricing subproblem. This may require large ng-memories, and thus advanced techniques are needed for solving the pricing subproblem in reasonable computational times. For example, Martinelli et al. (2014) described an algorithm based on the decremental state-space relaxation (DSSR) technique suggested by Righini and Salani (2008) where ng-memories are iteratively augmented while the best column generated in the pricing subproblem is not ng-feasible w.r.t some large ng-memories.

Good bounds can also be obtained if ng-memories are very well chosen, but not necessarily large. In this regard, Roberti and Mingozzi (2014) introduced the dynamic ng-path relaxation, which is roughly a sequence of non-decreasing lower bounds \( LB(\mathcal{M}_1), LB(\mathcal{M}_2), \ldots, LB(\mathcal{M}_k) \) associated with dynamically defined vertex-based ng-memories \( \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k \). Initial ng-memories \( \mathcal{M}_1 \) correspond to the \( \Delta^1 \) nearest customers, and memories \( \mathcal{M}_{t+1} \) are computed by extending memories \( \mathcal{M}_t, t \in \{1, \ldots, k-1\} \), in order to forbid the smallest cycle of the column with the largest primal value in a near-optimal solution of the linear relaxation of LP(\( \mathcal{M}_t \)). A limit of \( \Delta^2 \) is imposed for the size of each ng-memory, and the method stops when no cycle can be forbidden. The interested reader should consult (Roberti and Mingozzi, 2014) for further details.

In this section, we propose improvements to the dynamic ng-path, the main difference being the possibility to also reduce the ng-memories. This reduction is carried out if the pricing subproblems in the previous iteration of the method have been considered expensive. However, the role of this reduction is not only to make the pricing subproblem easier, but also to allow one a better choice of ng-memories. Finally, instead of forbidding cycles of a single column, we employ a potentially more aggressive algorithm for augmenting the ng-memories. The fully dynamic ng-path relaxation is outlined in Algorithm 2, and its main ingredients are described next.

Algorithm 3 presents the procedure adopted to augment ng-memories. Up to two phases are executed for each ng-route \( R \in \mathcal{R} \): first, the procedure attempts to forbid all cycles \( H \in \mathcal{H}(R) \) with at most \( \alpha \) vertices, where \( \mathcal{H}(R) \) denotes the set of all cycles of \( R \) (Phase 1); if no cycle could be forbidden in Phase 1, then all cycles \( H \in \mathcal{H}(R) \) are considered, regardless of size, and \( R \) is skipped after one cycle could be forbidden (Phase 2). We prioritize small cycles because they require less augmentations to be forbidden and many of them are likely to appear repeatedly in low-cost ng-routes. The ng-routes are sorted in a non-increasing order of primal value or non-decreasing order of reduced cost, depending on where the procedure is called from (Algorithm 2 or Algorithm 4, respectively). The procedure stops if \( \beta \) ng-routes have been explicitly forbidden. Notice that parameters \( \alpha \) and \( \beta \) control the aggressiveness of the ng-memories augmentation. The
Algorithm 2 Fully Dynamic \( ng \)-Path Relaxation

1: procedure fullyDynNgPath\( (\vec{M}, \Delta^{max}, \alpha_I, \beta_I, \alpha_R, \beta_R) \)
2: \( k \leftarrow 1, \vec{M}_1 \leftarrow \vec{M} \)
3: repeat
4: \( k \leftarrow 1, \vec{M}_1 \leftarrow \vec{M} \)
5: Compute \( (\bar{\lambda}_k, \bar{\pi}_k) \), an optimal primal-dual solution pair of \( LP(\vec{M}_k) \)
6: if \( ng \)-memories reduction condition is satisfied then
7: \( \vec{M}_k \leftarrow \text{reduceNGMemories}(\vec{M}_k, \bar{\pi}_k, \alpha_R, \beta_R) \)
8: Let \( R \) be the set of \( ng \)-routes associated with the primal solution \( \bar{\lambda}_k \)
9: \( \vec{M}_{k+1} \leftarrow \text{augmentNGMemories}(R, \vec{M}_k, \emptyset, \Delta^{max}, \alpha_I, \beta_I, \text{agressive}) \)
10: until stop condition is met

other parameters of this procedure — namely \( A \), \( \Delta^{max} \) and \( \text{mode} \) — will be explained later in this section.

Quickly, some augmentations performed in previous iterations may become unnecessary to guarantee the bound of the current iteration \( k \). Therefore, we propose Algorithm 4 to reduce \( ng \)-memories \( \vec{M}_k \). This algorithm is based on the same DSSR technique suggested by Righini and Salani (2008), and also implemented by Martinelli et al. (2014). A problem-dependent condition (pricing time, number of non-dominated labels, etc.) is used to trigger such reductions. Initially empty \( ng \)-memories are iteratively augmented in order to forbid the columns with the best reduced costs w.r.t \( \bar{\pi}_k \). Of course, those improvements are confined to \( \vec{M}_k \) and new iterations are performed as long as the best column generated is not \( ng \)-feasible w.r.t \( \vec{M}_k \). The algorithm ends up with a set of reduced \( ng \)-memories \( \vec{M} \) such that \( LB(\vec{M}) = LB(\vec{M}_k) \).

Even though the final \( ng \)-memories are not necessarily minimal, in practice we have observed that Algorithm 4 often reduces significantly the size of the \( ng \)-memories.

Remark that we have adopted the same algorithm for augmenting \( ng \)-memories (i) in the main loop of proposed relaxation, and (ii) inside the \( ng \)-memories reduction algorithm, but with different parameters. In case (i), more aggressive augmentations are needed for the sake of convergence, whereas in case (ii) moderate augmentations are performed in order to get smaller final \( ng \)-memories. This is mainly controlled by parameter \text{mode} of procedure forbidCycle(\( \cdot \)). Let us consider cycle \( H = (a_0 = (i_0 = v, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1} = v)) \). In moderate mode, only the cycles \( H = (v, i_1, \ldots, i_k, v) \), with \( k = 1, \ldots, p \), are explicitly forbidden, i.e., we add arcs \( a_0, \ldots, a_{p-1} \) to \( \vec{M}_v \). On the other hand, in aggressive mode, any cycle starting and ending at \( v \) and passing through a subset of \( \{i_1, \ldots, i_p\} \) is explicitly forbidden, hence we add any arc between nodes in \( \{v, i_1, \ldots, i_p\} \) to \( \vec{M}_v \) (see Algorithm 6). We should also point out that even though arc-based \( ng \)-memories are used, the complexity of the labeling algorithm is still somewhat vertex-dependent. As usual, let \( \delta^{-}(j) \) denote the set of arcs entering vertex \( j \) and let \( \gamma(j) \) be the number of \( ng \)-memories arcs \( \delta^{-}(j) \) belong to. The number of non-dominated labels representing paths ending at a vertex \( j \) may be exponential on \( \gamma(j) \). Thus, procedure cycleCanBeForbidden(\( \cdot \)),
Algorithm 3

\begin{algorithm}
\begin{algorithmic}[1]
\Procedure{augmentNgMemories}{\(\mathcal{R}, \hat{\mathcal{M}}, \mathcal{A}, \Delta^{max}, \alpha, \beta, \text{mode}\)}
\Input \(\mathcal{R}\): set of target \(ng\)-routes, \(\hat{\mathcal{M}}\): current \(ng\)-memories, \(\mathcal{A}\): set of forbidden augmentations, \(\Delta^{max}\): maximum value allowed for \(\gamma(\cdot)\), \(\alpha\): maximum cycle size in Phase 1, \(\beta\): maximum number of \(ng\)-routes explicitly forbidden, \text{mode}: mode of augmentation.
\Output new \(ng\)-memories
\State \(\mathcal{F}_H \leftarrow \emptyset, \mathcal{F}_R \leftarrow \emptyset\)
\For {each \(R \in \mathcal{R}\)}
\State phase $\leftarrow 1$
\For {each \(H = (v, \ldots, v) \in \mathcal{H}(R)\) in non-decreasing order of size}
\If {\(|H| > \alpha\) and phase $= 1$}
\If {\(\mathcal{H}(R) \cap \mathcal{F}_H = \emptyset\)}
\State phase $\leftarrow 2$
\Else
\State break
\EndIf
\EndIf
\EndFor
\If {cycleCanBeForbidden\(\(H, \hat{\mathcal{M}}, \Delta^{max}, \mathcal{A}\)\)}
\State \(\hat{\mathcal{M}} \leftarrow \text{forbidCycle}(H, \hat{\mathcal{M}}, \text{mode})\)
\State \(\mathcal{F}_H \leftarrow \mathcal{F}_H \cup \{H\}, \mathcal{F}_R \leftarrow \mathcal{F}_R \cup \{R\}\)
\If {phase $= 2$}
\State break
\EndIf
\EndIf
\EndFor
\If {\(|\mathcal{F}_R| \geq \beta\)}
\State stop
\EndIf
\Return \(\hat{\mathcal{M}}\)
\EndProcedure
\end{algorithmic}
\end{algorithm}

The algorithm described in Algorithm 5, considers a maximum value \(\Delta^{max}\) allowed for \(\gamma(\cdot)\). Additionally, it also considers a set \(\mathcal{A}\) of forbidden augmentations, which is used in the context of the \(ng\)-memories reduction algorithm.

4. Branch-and-Price Algorithm

In this section, we describe the proposed branch-and-price algorithm (BP) for MLP. The solution of a node in our algorithm, outlined in Algorithm 7, is an iterative approach based on the fully dynamic \(ng\)-path relaxation described in Section 3.3. At each iteration \(k\), we first solve \(LP(\hat{\mathcal{M}}_k)\) by means of a two-stage column generation. Of course, branching constraints may also be present in this linear program. In Stage 1, dominance tests take into account only conditions (I), (II) and (III), thus a single \(ng\)-path is kept for a given vertex \(i\) and resource consumption \(w\), which is the one with minimum reduced cost. This is a heuristic pricing intended to quickly generate good \(ng\)-paths. When Stage 1 fails to find a \(ng\)-path with negative reduced cost, we switch to Stage 2, where the exact pricing is solved. In both phases, the dual stabilization technique of Pessoa et al. (2013) is applied for the sake of convergence. Stage 1 (2) is solved by a mono-directional (bidirectional) labeling algorithm that returns at most 50 (300) \(ng\)-paths. Details on
Algorithm 4 \textit{ng}-Memories Reduction Algorithm

\begin{algorithm}
\begin{algorithmic}
\State \textbf{procedure} reduceNGMemories($\vec{M}$, $\pi^*$, $\alpha$, $\beta$)
\State \textbf{input:} $\vec{M}$: current \textit{ng}-memories, $\pi^*$: dual solution of $LP(\vec{M})$, and parameters to augmentNgMemories($\cdot$)
\State \textbf{output:} new \textit{ng}-memories
\State $\vec{M}_{ori} \leftarrow \vec{M}$, $\vec{M} \leftarrow \emptyset$, \textit{ng}-feasible $\leftarrow$ false
\While {not \textit{ng}-feasible}
\State $R \leftarrow \text{LabelingAlgorithm}(\vec{M}, \pi^*)$
\If {best route in $R$ is \textit{ng}-feasible w.r.t $\vec{M}_{ori}$}
\State \textit{ng}-feasible $\leftarrow$ true
\Else
\State Define $\mathcal{A}$ as the set of augmentations that are not confined to $\vec{M}_{ori}$
\State $\vec{M} \leftarrow \text{augmentNgMemories}(R, \vec{M}, \mathcal{A}, \infty, \alpha, \beta, \text{moderate})$
\EndIf
\EndWhile
\State \textbf{return} $\vec{M}$
\end{algorithmic}
\end{algorithm}

such algorithm will be given in Section 4.1.

If the node is the root, the first memories $\vec{M}_1$ are equivalent to \textit{ng}-sets of size 8 defined according to the classical distance-based rule — time-based for the case of MLP. Otherwise, they correspond to the final memories of the parent node, which are inherited by the child. In any node, a hybrid strategy for augmenting \textit{ng}-memories may be used. Initially, we set mode $\leftarrow$ aggressive.

The method switches to moderate mode if the computational time of a single call to the labeling algorithm exceeds a threshold value $t_{\text{red}} = 100$ seconds. In this case, column generation is interrupted and the \textit{ng}-memories reduction algorithm (see Section 3.3) is called with $\alpha_R = 5$ and $\beta_R = 200$; afterwards, the current iteration is restarted with the new augmentation mode.

Regardless of mode, we have adopted $\alpha_I = 5$, $\beta_I = 200$ and $\Delta_{\text{max}} = 63$.

As we have already discussed, the pricing problem corresponds to finding a least-cost \textit{ng}-path in the resource constrained network $G'$ defined in Section 2. However, in practice, we work with an extended network $G^{\text{ext}} = (V^{\text{ext}}, A^{\text{ext}})$, where:

\begin{align*}
V^{\text{ext}} &= \{(i, w) : i, w \in \{1, \ldots, n\} \cup \{(s, 0), (t, n + 1)\} \\
A^{\text{ext}} &= \{((i, w), (j, w + 1)) : (i, j) \in A, (i, w) \in V^{\text{ext}}, (j, w + 1) \in V^{\text{ext}}\}
\end{align*}

Network $G^{\text{ext}}$ is defined in such a way that resource constraints are naturally satisfied by any \textit{ng}-path. Once an optimal dual solution $\bar{\pi}_k$ is available, a reduced cost fixing procedure is used to remove from $A^{\text{ext}}$ the arcs that can not participate in a solution that improves the current upper bound. For fixing an arc $((i, w), (j, w + 1))$, one should prove that the minimum reduced cost of a \textit{ng}-path traversing this arc is above a given threshold. To compute this cost, the minimum reduced cost of a partial path ending at $(i, w)$ and of a partial path from $(j, w + 1)$ to the sink node
Algorithm 5 Test for Cycle Elimination

1: procedure cycleCanBeForbidden($H, \overrightarrow{M}, A, \Delta^{max}$)
2:  
3:  
4:  
5:  
6:  
7:  
8:  
9:  
10: return true

Algorithm 6 Cycle Elimination

1: procedure forbidCycle($H, \overrightarrow{M}, \text{mode}$)
2:  
3:  
4:  
5:  
6:  
7:  
8:  
9: return true

are computed by a forward and a backward labeling algorithms, respectively. Such a procedure is well-known in the literature and have been applied in many routing problems (see, for instance, (Roberti and Mingozzi, 2014; Pecin et al., 2016)). Dual solution $\overline{\pi}_k$ is also used in an enumerative procedure that tries to finish the node. As implemented by Roberti and Mingozzi (2014), the enumeration is performed by a mono-directional labeling that computes only elementary paths, using completion bounds associated with $ng$-paths to prune unpromising partial paths. However, we abort the enumeration if the number of non-dominated labels is greater than 10 millions. If enumeration finishes, either a single path representing the best integer solution for the node is returned, or all labels are dominated, meaning that this solution does not improve the current upper bound.

Besides the obvious stop conditions (node is solved or pruned), Algorithm 7 stops if:

- **Stop condition I**: Labeling algorithm time has exceeded the threshold value $t^\text{red}$ twice.
- **Stop condition II**: No column could be forbidden by $\text{augmentNgMemories}()$ because of
Algorithm 7 Solution of a node

1:   \textbf{procedure} solveNode($\mathcal{M}, B$)
2:   \hspace{1em} $k \leftarrow 1, \mathcal{M}_1 \leftarrow \mathcal{M}$
3:   \hspace{1em} mode $\leftarrow$ aggressive
4:   \hspace{1em} repeat
5:      Compute ($\bar{\lambda}_k, \bar{\pi}_k$), an optimal primal-dual solution pair of $LP(\mathcal{M}_k) +$ branching constraints $B$
6:      \hspace{1em} if \textit{ng}-memories reduction condition is satisfied then
7:         \hspace{2em} $\mathcal{M}_k \leftarrow$ reduceNGMemories($\mathcal{M}_k, \bar{\pi}_k, 5, 200$)
8:         \hspace{2em} mode $\leftarrow$ moderate
9:         \hspace{2em} continue
10:     \hspace{1em} Apply reduced cost fixing
11:     \hspace{1em} Try to finish the node by enumeration
12:     \hspace{1em} Let $\mathcal{R}$ be the set of \textit{ng}-routes associated with the primal solution $\bar{\lambda}_k$
13:     \hspace{1em} $\mathcal{M}_{k+1} \leftarrow$ augmentNgMemories($\mathcal{R}, \mathcal{M}_k, \emptyset, 63, 5, 200$, mode)
14:     \hspace{1em} $k \leftarrow k + 1$
15:   \hspace{1em} until stop condition is met

\[ \Delta^{max}. \]

- \textbf{Stop condition III:} For 5 times, the gap of the current iteration is less than 2\% smaller than the one of the previous iteration.

We branch on an edge \{i, j\} defined by a 3-phase strong branching mechanism in the spirit of the works of Røpke (2012) and Pecin et al. (2016). Hence, vertices i and j must appear consecutively (either i $\rightarrow$ j or j $\rightarrow$ i) in the solution of one child and must not be consecutive in the solution of the other child. In Phase 0, we define $\min\{100, TS(v)\}$ candidate edges, where $TS(v)$ is an estimation for the size of the tree rooted at node $v$. Such estimation takes into account the average bound improvement in the branch history, following the model of Kullmann (2009). $TS(v) = \infty$ if $v$ is the root node. First, we select up to half of the edges from a pool containing the candidates evaluated in previous executions of Phase 2 in the whole branch history. The other candidates are the edges whose values are the closest to 0.5 in the current fractional solution. For each candidate selected in Phase 0, we perform a rough evaluation of both children by solving the master LP of node $v$ with the corresponding branching constraint, but without any column generation. This is Phase 1, where the best $\min\{5, TS(v)/10\}$ candidates are selected, according to the product rule of (Achterberg, 2007), to go to the last phase. Phase 2 uses the same approach as for Phase 1, but heuristic column generation (Stage 1) is applied when evaluating the children. The selected edge is the one with the best score in Phase 2, also according to the product rule.

We now provide more information on the implementations of the labeling algorithm and of MPLD.
4.1. Labeling Algorithm

Following many related works in the literature — for instance, (Martinelli et al., 2014) and (Pecin et al., 2016) — we have adopted the concept of bucket in our implementation of the general method described in Algorithm 1. A forward bucket $B(j, w)$ is a data structure that stores all non-dominated labels associated with forward $s-j$ paths with resource consumption $w$. Therefore, for each vertex $j$ we define buckets $B(j, w), \forall w \in \{l_j, \ldots, u_j\}$. To accelerate the algorithm, labels in a bucket are kept sorted by non-decreasing order of cost and dominance checks are performed only between labels of the same bucket, just as implemented by Pecin et al. (2016). Buckets are considered by the algorithm in a non-decreasing order of resource consumption. When a bucket $B(j, w)$ is reached, we extend all labels in $B(j, w)$ over all arcs $(j, k) \in A$. At the end, the optimal $s-t$ paths will be stored in the buckets associated with the sink node $t$.

The algorithm outlined above is called mono-directional since all labels kept correspond to partial paths from the source node to some node $j$ and arcs are traversed in their regular directions. However, in our implementation we have used a bidirectional labeling algorithm, which generates more diversified set of $s-t$ paths and is typically faster than its mono-directional counterpart. A backward path corresponds to a partial path from the sink to some node $j$ obtained by traversing arcs in their reverse directions. Labels corresponding to such paths are stored in backward buckets $\hat{B}(j, w)$. In the forward (backward) labeling algorithm, one computes non-dominated labels for buckets $B(j, w)$ ($\hat{B}(j, w)$) such that $w \leq w^*$ ($w \geq w^*$), where $w^*$ is a threshold value that ideally should be defined so as to balance the computational effort of the forward and the backward labeling algorithms. Then, a concatenation procedure is executed to build complete $s-t$ paths from the partial forward and backward paths. The reader is referred to (Righini and Salani, 2006) for further references on bidirectional labeling.

4.2. Multiple Partial Label Dominance

Let us consider again labels $L(P')$ and $L(P)$ such that conditions stated in Proposition 1 are satisfied, and define $\Pi' = \Pi(P') \setminus \Pi(P)$. As we have discussed, the extensions to vertices $j \in V \setminus \bigcup_{k \in \Pi'} M_k$ can be avoided for label $L(P)$. For example, in Figure 1, one can verify that any extension to vertices $j \notin M_3$ can be avoided for label $L(P_4)$ because of label $L(P_3)$, and thus set $\xi(P_4)$ is increased as $\xi(P_4) \leftarrow \xi(P_4) \cup \{2, 5, 6\}$. Such an operation is performed several times during the course of the labeling algorithm and an efficient implementation is required, otherwise the gains incurred by MPLD will not pay off. In what follows, we will describe two approaches that we have tried to take advantage of this new dominance rule.

4.2.1. Explicit Representation of $\xi(\cdot)$

We first tried an explicit representation of sets $\xi(\cdot)$ inside each label $L$.

- **Bitmap representation:** Sets $\xi(\cdot)$ are represented as bitmaps implemented over 64-bit integers. For each vertex $i \in V$, MPLD can avoid the extension of labels $L(P) \in U_i$ only
over a set \( E_i \) of at most 63 arcs. This set is composed of arcs \((i, j) \in A\) with the smallest costs, and the 64th bit of the bitmaps are always set to zero to indicate that arcs \((i, j) \notin E_i\) are not handled by MPLD. In practice, 63 is a reasonable number of arcs tracked and allows very quick operations since a single 64-bit integer is needed to represent a bitmap. The bit corresponding to arc \((i, j)\) is checked in constant time through bitwise operations before extending labels \(L(P) \in U_i\) over \((i, j)\).

- **Precomputed union of memories:** The number of different possible sets \(\Pi(P)\) for a path \(P\) ending at vertex \(i\) is \(2^{|N_i|}\). Thus, for two labels \(L(P_1)\) and \(L(P_2)\) such that \(v(P_1) = v(P_2) = i\), the number of different possible sets \(\Pi' = \Pi(P_1) \setminus \Pi(P_2)\) is also \(2^{|N_i|}\). For reasonable values of \(|N_i|\), it is practical to precompute sets \(V \setminus \bigcup_{k \in \Pi'} M_k\) for every possible \(\Pi'\). In fact, we have observed that gains of MPLD are diminished if these sets are not precomputed. When testing the dominance of label \(L(P_2)\) by label \(L(P_1)\), a bitmap representing \(V \setminus \bigcup_{k \in \Pi'} M_k\) is retrieved and used to update \(\xi(P_2)\) in constant time through bitwise operations. Notice that in our implementation such dominance checks are performed within a same bucket.

Computational experiments showed that the computational time speed-up incurred by this approach is not significant. The main limitation is that one needs to restrict the cardinality of sets \(\xi(\cdot)\) to at most 63 in order to quickly manipulate the bitmaps representing them. In this case, MPLD is not completely explored. Moreover, we have adopted a distance-based criterion to decide the 63 extensions tracked, which may be a very crude criterion in some cases. For example, an extension over a long arc (i.e., an arc connecting two customers that are far away from each other) will probably generate a label \(L(P)\) with a large cost, but with few customers in set \(\Pi(P)\). This latter attribute makes very hard to dominate label \(L(P)\), increasing the number of dominance checks in the destination bucket. A dynamic definition of tracked extensions would probably improve results, but the following approach is simpler and mitigates all aforementioned problems.

### 4.2.2. Implicit Representation of \(\xi(\cdot)\)

Recall that, in our implementation of the labeling algorithm, buckets are considered in a predefined order and labels of a same bucket are extended in a non-decreasing order of cost. Moreover, we extend all labels of a bucket over an arc, then over another arc and so on. Let us consider a (forward or backward) bucket \(B(i, w)\) and an arc \((i, j)\). Suppose that a label \(L(P') \in B(i, w)\) has already been extended to a label \(L(P' + j)\). Now let \(L(P) \in B(i, w)\) be a label that has not yet been extended over \((i, j)\) such that \(\Pi(P' + j) \subseteq \Pi(P + j)\). Since its cost is larger, \(L(P + j)\) will be dominated by label \(L(P' + j)\) and thus one can avoid the extension of \(L(P)\) over \((i, j)\) — this is just an example of MPLD. In general, for each bucket \(B(i, w)\) and arc \((i, j)\), we keep a list of bitmaps representing sets \(\Pi(\cdot)\) of already extended labels. Then, before extending a label \(L(P) \in B(i, w)\) over \((i, j)\), we first compute \(\Pi(P + j)\) and check if it is a superset of some \(\Pi(P' + j)\) contained in the list. If so, the extension is not necessary and we proceed to
If \( \Pi(P + j) = \{j\} \), then we can stop extending labels in \( B(i, w) \) over \((i, j)\). Furthermore, there is no need to keep the list after considering the extension of all labels in \( B(i, w) \) over \((i, j)\). This approach better explores the potential of MPLD and typically yields a significant speed up in the pricing time, as will be seen in next section.

5. Computational Experiments

This section reports our computational experiments over 40 TSPLIB instances with up to 200 vertices. The proposed BP algorithm was implemented in C++ over the BaPCod platform of (Vanderbeck et al., 2017). The LP solver adopted is IBM CPLEX Optimizer version 12.6.0. All experiments were conducted on an Intel Xeon E5-2680 v3, running at 2.5 GHz with a single thread. We will compare our results to those obtained by the algorithm of Roberti and Mingozzi (2014) on an Intel Xeon X7350, running at 2.93 GHz. According to the CPU benchmark website www.cpuboss.com, the single thread performance of our CPU is about 1.5 times better than the one of Roberti and Mingozzi (2014). Therefore, our computational times are increased by this factor in Table 1.

5.1. Main Results

Table 1 presents the main results of the proposed BP and the results of Roberti and Mingozzi (2014) over instances with up to 152 vertices. In this case, we set a time limit of 2 days for BP. For what concerns the results of Roberti and Mingozzi (2014), we report the following data: \( lb \), the final lower bound; \( t_{lb} \), the computational time to obtain such lower bound; \( t_{tot} \), the total computational time; and gap, the percentage gap before applying enumeration. A symbol “-” in column \( t_{tot} \) indicates that enumeration failed to find the optimal solution. The results of the proposed method are divided into root node and complete BP. For the root node, we report the first and last lower bounds obtained, \( lb^0 \) and \( lb^k \), where \( k \) is the number of \( ng \)-memories augmentations performed. Further, we present the average and maximum values of \( \gamma(\cdot) \) and the time spent at the root node. We also indicate if the method has switched to moderate mode in the root node. For the complete BP, we report the final bounds and gap, the total computational time and the number of nodes solved. All computational times in Table 1 are given in seconds.

We can observe in Table 1 that the proposed BP outperforms the method of Roberti and Mingozzi (2014) in most instances. In particular, 6 instances were solved for the first time and the BKS for instance kroA150 was improved from 1831766 to the optimal value 1825769. The main advantage of our method is the possibility of dealing with larger \( ng \)-memories without combinatorial explosion. For example, for the 6 instances solved only by BP, the maximum values of \( \gamma(\cdot) \) are greater than 30, reaching 62 in eil101. This is possible only because of the proposed generalized definition of \( ng \)-sets in terms of arcs. Moreover, our computational experience shows that the hybridization of aggressive and moderate modes is crucial for the robustness
Table 1: Results of BP over TSPLIB instances used by Roberto and Mingozzi (2014). Computational times are normalized according to the CPU benchmark website www.cpuboss.com.

<table>
<thead>
<tr>
<th>Instance</th>
<th>( z^* )</th>
<th>lb</th>
<th>( t^* )</th>
<th>( t_{tot} )</th>
<th>( \text{gap} )</th>
<th>( \text{Root Node} )</th>
<th>( \text{Proposed Method} )</th>
<th>( \text{B&amp;P} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>dantzig42</td>
<td>12528</td>
<td>12528</td>
<td>5</td>
<td>5</td>
<td>0.00</td>
<td>12528</td>
<td>12528</td>
<td>0.00 1 4.7</td>
</tr>
<tr>
<td>swiss42</td>
<td>22327</td>
<td>22327</td>
<td>3</td>
<td>3</td>
<td>0.00</td>
<td>22327</td>
<td>22327</td>
<td>0.00 1 2.1</td>
</tr>
<tr>
<td>att48</td>
<td>209320</td>
<td>209320</td>
<td>7</td>
<td>7</td>
<td>0.00</td>
<td>209320</td>
<td>209320</td>
<td>0.00 1 8.5</td>
</tr>
<tr>
<td>gr48</td>
<td>102378</td>
<td>102378</td>
<td>15</td>
<td>15</td>
<td>0.00</td>
<td>102378</td>
<td>102378</td>
<td>0.00 1 7.2</td>
</tr>
<tr>
<td>hk48</td>
<td>247926</td>
<td>247926</td>
<td>19</td>
<td>19</td>
<td>0.00</td>
<td>247926</td>
<td>247926</td>
<td>0.00 1 10.1</td>
</tr>
<tr>
<td>eil51</td>
<td>10178</td>
<td>10178</td>
<td>10</td>
<td>10</td>
<td>0.00</td>
<td>10178</td>
<td>10178</td>
<td>0.00 1 6.5</td>
</tr>
<tr>
<td>berlin52</td>
<td>143721</td>
<td>143721</td>
<td>10</td>
<td>10</td>
<td>0.00</td>
<td>143721</td>
<td>143721</td>
<td>0.00 1 17.7</td>
</tr>
<tr>
<td>brazil58</td>
<td>512361</td>
<td>512361</td>
<td>36</td>
<td>36</td>
<td>0.00</td>
<td>512361</td>
<td>512361</td>
<td>0.00 1 17.7</td>
</tr>
<tr>
<td>st70</td>
<td>20557</td>
<td>20557</td>
<td>61</td>
<td>61</td>
<td>0.00</td>
<td>20557</td>
<td>20557</td>
<td>0.00 1 24.0</td>
</tr>
<tr>
<td>eil76</td>
<td>17976</td>
<td>17976</td>
<td>91</td>
<td>91</td>
<td>0.00</td>
<td>17976</td>
<td>17976</td>
<td>0.00 1 26.6</td>
</tr>
<tr>
<td>pr76</td>
<td>345524</td>
<td>345524</td>
<td>169</td>
<td>223</td>
<td>0.93</td>
<td>345524</td>
<td>345524</td>
<td>0.00 1 193.0</td>
</tr>
<tr>
<td>gr96</td>
<td>2097170</td>
<td>2097170</td>
<td>365</td>
<td>369</td>
<td>0.45</td>
<td>2097170</td>
<td>2097170</td>
<td>0.00 1 215.3</td>
</tr>
<tr>
<td>rat99</td>
<td>57986</td>
<td>57986</td>
<td>514</td>
<td>576</td>
<td>0.77</td>
<td>57986</td>
<td>57986</td>
<td>0.00 3 3180.4</td>
</tr>
<tr>
<td>kroA100</td>
<td>988098</td>
<td>988098</td>
<td>679</td>
<td>688</td>
<td>0.27</td>
<td>988098</td>
<td>988098</td>
<td>0.00 1 337.2</td>
</tr>
<tr>
<td>kroB100</td>
<td>969008</td>
<td>969008</td>
<td>197</td>
<td>197</td>
<td>0.00</td>
<td>969008</td>
<td>969008</td>
<td>0.00 1 75.2</td>
</tr>
<tr>
<td>kroC100</td>
<td>961324</td>
<td>961324</td>
<td>409</td>
<td>428</td>
<td>0.39</td>
<td>961324</td>
<td>961324</td>
<td>0.00 1 815.9</td>
</tr>
<tr>
<td>kroD100</td>
<td>976965</td>
<td>976965</td>
<td>708</td>
<td>729</td>
<td>0.42</td>
<td>976965</td>
<td>976965</td>
<td>0.00 1 290.0</td>
</tr>
<tr>
<td>kroE100</td>
<td>971266</td>
<td>971266</td>
<td>266</td>
<td>266</td>
<td>0.00</td>
<td>971266</td>
<td>971266</td>
<td>0.00 1 87.7</td>
</tr>
<tr>
<td>rd100</td>
<td>340047</td>
<td>340047</td>
<td>317</td>
<td>317</td>
<td>0.04</td>
<td>340047</td>
<td>340047</td>
<td>0.00 1 213.3</td>
</tr>
<tr>
<td>eil101</td>
<td>27513</td>
<td>27513</td>
<td>630</td>
<td>1.00</td>
<td>26898</td>
<td>27319</td>
<td>27319</td>
<td>0.00 3 6238.9</td>
</tr>
<tr>
<td>lin105</td>
<td>603910</td>
<td>603910</td>
<td>154</td>
<td>154</td>
<td>0.00</td>
<td>603910</td>
<td>603910</td>
<td>0.00 1 85.5</td>
</tr>
<tr>
<td>pr107</td>
<td>2026626</td>
<td>2026626</td>
<td>246</td>
<td>246</td>
<td>0.00</td>
<td>2007255</td>
<td>2026626</td>
<td>0.00 1 470.9</td>
</tr>
<tr>
<td>gr120</td>
<td>363454</td>
<td>363454</td>
<td>4685</td>
<td>0.82</td>
<td>353054</td>
<td>359970</td>
<td>359970</td>
<td>0.00 11 79351.6</td>
</tr>
<tr>
<td>pr124</td>
<td>3154346</td>
<td>3154346</td>
<td>14080</td>
<td>14080</td>
<td>0.00</td>
<td>3001043</td>
<td>3154346</td>
<td>0.00 1 58283.6</td>
</tr>
<tr>
<td>bier12</td>
<td>4545005</td>
<td>4545005</td>
<td>723</td>
<td>723</td>
<td>0.00</td>
<td>4513768</td>
<td>4545005</td>
<td>0.00 1 307.7</td>
</tr>
<tr>
<td>ch130</td>
<td>348974</td>
<td>348974</td>
<td>2385</td>
<td>0.53</td>
<td>341809</td>
<td>348013</td>
<td>348013</td>
<td>0.00 3 7810.0</td>
</tr>
<tr>
<td>pr136</td>
<td>6199268</td>
<td>6199268</td>
<td>17190</td>
<td>0.63</td>
<td>5908131</td>
<td>6155720</td>
<td>6155720</td>
<td>0.39 7 259200.0</td>
</tr>
<tr>
<td>gr137</td>
<td>4061498</td>
<td>4061498</td>
<td>7844</td>
<td>0.89</td>
<td>3952022</td>
<td>4017150</td>
<td>4017150</td>
<td>0.00 15 65614.2</td>
</tr>
<tr>
<td>pr144</td>
<td>384137</td>
<td>384137</td>
<td>8043</td>
<td>0.11</td>
<td>3671878</td>
<td>3846137</td>
<td>3846137</td>
<td>0.00 1 6101.6</td>
</tr>
<tr>
<td>ch150</td>
<td>444424</td>
<td>444424</td>
<td>2585</td>
<td>0.08</td>
<td>438450</td>
<td>444424</td>
<td>444424</td>
<td>0.00 1 1743.0</td>
</tr>
<tr>
<td>kroA150</td>
<td>1831766</td>
<td>1831766</td>
<td>5187</td>
<td>0.75</td>
<td>1794351</td>
<td>1813870</td>
<td>1813870</td>
<td>0.00 15 93566.8</td>
</tr>
<tr>
<td>kroB150</td>
<td>179204</td>
<td>179204</td>
<td>20120</td>
<td>0.96</td>
<td>1720498</td>
<td>1752709</td>
<td>1752709</td>
<td>0.00 19 259200.0</td>
</tr>
<tr>
<td>pr152</td>
<td>5064566</td>
<td>5064566</td>
<td>31218</td>
<td>2.63</td>
<td>4803249</td>
<td>4984428</td>
<td>4984428</td>
<td>0.00 3 259200.0</td>
</tr>
</tbody>
</table>
of proposed method. Even though some instances are better solved if only slower arc-based \textit{ng}-memories augmentations are performed (e.g. eil101 and gr120), others instances, such as pr124, could not be solved without \textbf{aggressive} mode. Instances pr136, pr152 and kroB150 could not be solved, but we remark that “pr” instances have a special structure with a lot of symmetry (see Figure 2). However, we will show in the next section that a specific parameterization of our method is capable of solving those 3 instances.

![Figure 2: Optimal solution of pr136.](image)

Table 2 shows the results of BP over large TSPLIB instances with up to 200 vertices. To our knowledge, this is the first time that such large instances are considered by an exact method for MLP. The upper bounds were computed by the method of Silva et al. (2012). Instances u159, si175, brg180 and rat195 were solved in reasonable computational times and small BP trees, but BP finished with considerable gaps for the other instances. For rat195, BP found an optimal solution with cost 218632, an improvement of 43 units over the heuristic solution found by the method of Silva et al. (2012). In spite of the new contributions presented in this paper, MLP instances with about 200 vertices still seem to be very challenging.

5.2. \textit{Longer runs with moderate mode}

Here we show that our method can solve some more hard instances by using a different parameterization. The idea of this parameterization is to augment \textit{ng}-memories very slowly and over a large number of iterations. More precisely, we adopt \textbf{moderate} mode in all iterations of Algorithm 7, set a time limit of 5 days for BP and change stop conditions I and III as below.
Table 2: Results of BP over large TSPLIB instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>$z^*_l$</th>
<th>$lb^0$</th>
<th>$lb^k$</th>
<th>mode</th>
<th>Max. $\gamma(-)$</th>
<th>Avg. $\gamma(-)$</th>
<th>$t_{root}$</th>
<th>$lb$</th>
<th>$ub$</th>
<th>gap</th>
<th>nodes</th>
<th>$t_{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>u159</td>
<td>2972030</td>
<td>2892904</td>
<td>2960764</td>
<td>25</td>
<td>yes</td>
<td>43</td>
<td>26.65</td>
<td>19881.7</td>
<td>2972030</td>
<td>2972030</td>
<td>0.00</td>
<td>3</td>
</tr>
<tr>
<td>si175</td>
<td>1808532</td>
<td>1808160</td>
<td>1808195</td>
<td>20</td>
<td>yes</td>
<td>38</td>
<td>22.04</td>
<td>22874.3</td>
<td>1808532</td>
<td>1808532</td>
<td>0.00</td>
<td>3</td>
</tr>
<tr>
<td>brg180</td>
<td>174750</td>
<td>164762</td>
<td>174750</td>
<td>8</td>
<td>no</td>
<td>18</td>
<td>13.42</td>
<td>3353.6</td>
<td>174750</td>
<td>174750</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>rat195</td>
<td>218675</td>
<td>216725</td>
<td>217830</td>
<td>13</td>
<td>yes</td>
<td>50</td>
<td>25.50</td>
<td>3529.3</td>
<td>218675</td>
<td>218675</td>
<td>0.00</td>
<td>11</td>
</tr>
<tr>
<td>si175</td>
<td>1186049</td>
<td>1128131</td>
<td>1144047</td>
<td>12</td>
<td>yes</td>
<td>23</td>
<td>10.97</td>
<td>30475.7</td>
<td>1147364</td>
<td>1186050</td>
<td>3.37</td>
<td>7</td>
</tr>
<tr>
<td>brg180</td>
<td>174750</td>
<td>164762</td>
<td>174750</td>
<td>8</td>
<td>no</td>
<td>18</td>
<td>13.42</td>
<td>3353.6</td>
<td>174750</td>
<td>174750</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>pr136</td>
<td>6199268</td>
<td>5908131</td>
<td>6182837</td>
<td>125</td>
<td>52</td>
<td>37.43</td>
<td>45071.2</td>
<td>6199268</td>
<td>6199268</td>
<td>0.00</td>
<td>15</td>
<td>187356.3</td>
</tr>
<tr>
<td>kroB150</td>
<td>1793204</td>
<td>1720498</td>
<td>1760182</td>
<td>103</td>
<td>63</td>
<td>44.98</td>
<td>13371.7</td>
<td>178654</td>
<td>1786546</td>
<td>0.00</td>
<td>23</td>
<td>293479.1</td>
</tr>
<tr>
<td>pr152</td>
<td>5064566</td>
<td>4803249</td>
<td>4984947</td>
<td>110</td>
<td>26</td>
<td>17.22</td>
<td>38839.8</td>
<td>506456</td>
<td>506456</td>
<td>0.00</td>
<td>3</td>
<td>281061.9</td>
</tr>
<tr>
<td>d198</td>
<td>1186049</td>
<td>1128131</td>
<td>1145300</td>
<td>101</td>
<td>31</td>
<td>17.68</td>
<td>27433.4</td>
<td>1152702</td>
<td>1186050</td>
<td>2.89</td>
<td>17</td>
<td>432000.0</td>
</tr>
<tr>
<td>kroA200</td>
<td>2672437</td>
<td>2600209</td>
<td>2631523</td>
<td>15</td>
<td>yes</td>
<td>45</td>
<td>24.39</td>
<td>20536.9</td>
<td>2637610</td>
<td>2672438</td>
<td>0.00</td>
<td>11</td>
</tr>
<tr>
<td>kroB200</td>
<td>2669515</td>
<td>2580374</td>
<td>2636041</td>
<td>15</td>
<td>yes</td>
<td>30</td>
<td>15.08</td>
<td>13806.2</td>
<td>2643616</td>
<td>2669516</td>
<td>0.98</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 3: Results of BP over hard instances with a different parameterization

<table>
<thead>
<tr>
<th>Instance</th>
<th>$z^*_l$</th>
<th>$lb^0$</th>
<th>$lb^k$</th>
<th>mode</th>
<th>Max. $\gamma(-)$</th>
<th>Avg. $\gamma(-)$</th>
<th>$t_{root}$</th>
<th>$lb$</th>
<th>$ub$</th>
<th>gap</th>
<th>nodes</th>
<th>$t_{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr136</td>
<td>6199268</td>
<td>5908131</td>
<td>6182837</td>
<td>125</td>
<td>52</td>
<td>37.43</td>
<td>45071.2</td>
<td>6199268</td>
<td>6199268</td>
<td>0.00</td>
<td>15</td>
<td>187356.3</td>
</tr>
<tr>
<td>kroB150</td>
<td>1793204</td>
<td>1720498</td>
<td>1760182</td>
<td>103</td>
<td>63</td>
<td>44.98</td>
<td>13371.7</td>
<td>178654</td>
<td>1786546</td>
<td>0.00</td>
<td>23</td>
<td>293479.1</td>
</tr>
<tr>
<td>pr152</td>
<td>5064566</td>
<td>4803249</td>
<td>4984947</td>
<td>110</td>
<td>26</td>
<td>17.22</td>
<td>38839.8</td>
<td>506456</td>
<td>506456</td>
<td>0.00</td>
<td>3</td>
<td>281061.9</td>
</tr>
<tr>
<td>d198</td>
<td>1186049</td>
<td>1128131</td>
<td>1145300</td>
<td>101</td>
<td>31</td>
<td>17.68</td>
<td>27433.4</td>
<td>1152702</td>
<td>1186050</td>
<td>2.89</td>
<td>17</td>
<td>432000.0</td>
</tr>
<tr>
<td>kroA200</td>
<td>2672437</td>
<td>2600209</td>
<td>2631523</td>
<td>15</td>
<td>yes</td>
<td>45</td>
<td>24.39</td>
<td>20536.9</td>
<td>2637610</td>
<td>2672438</td>
<td>0.00</td>
<td>11</td>
</tr>
<tr>
<td>kroB200</td>
<td>2669515</td>
<td>2580374</td>
<td>2636041</td>
<td>15</td>
<td>yes</td>
<td>30</td>
<td>15.08</td>
<td>13806.2</td>
<td>2643616</td>
<td>2669516</td>
<td>0.98</td>
<td>13</td>
</tr>
</tbody>
</table>

- **Stop condition I**: Labeling algorithm time has exceeded the threshold value $t_{red} = 150$.

- **Stop condition III**: For 100 times, the gap of the current iteration is less than 2% smaller than the one of the previous iteration.

Table 3 shows the results obtained. For the first time, instances pr136, pr152 and kroB150 were solved to optimality. Therefore, we solved all instances that were not solved by Roberto and Mingozzi (2014). Furthermore, the BKS for kroB150 was improved from 1793204 to the optimal value 1786546, an improvement of 0.03%. Still, we should point out that such a parameterization is useful only for hard instances. In general, BP has a worse performance if a weaker tailing off condition is used.

5.3. Multiple Partial Label Dominance

Table 4 shows the performance of the labeling algorithm with and without MPLD. The average results presented in that table concern the exact calls to the labeling algorithm in the first descent of column generation with $ng$-sets of a given fixed size in $\{8, 12, 16\}$. A representative subset of the instances was used. With $ng$-sets of size 16, instances pr124, pr144 and pr152 exceeded a time limit set for this experiment, thus no results are reported in those cases. It can be seen in Table 4 that the new dominance rule has a significant impact on the labeling algorithm. The average number of extensions decreased by an order of magnitude for any tested instance. This allowed the algorithm to achieve remarkable average speedups of 4.37, 6.01 and 7.28 in computational
time for \( ng \)-sets of sizes 8, 12 and 16, respectively. Of course, with good lower and upper bounds and some rounds of reduced cost fixing, the pricing networks gets sparser and those speedups are smaller, but still considerable. Surprisingly, for most instances, the percentage of non-dominated labels that are never extended — column MPLD (%) — is significantly large. This means that not rarely a label is completely dominated by a set of two or more labels, but not by a single label as required by the classical dominance rule.

| Instance | \( |N_i| \) | Avg. time (s) | Avg. number of extensions | MPLD (%) |
|----------|-----------|---------------|--------------------------|----------|
|          | without   | with factor   | without (\( \times 10^6 \)) | with (\( \times 10^6 \)) factor |          |
| gr120    | 8         | 1.16          | 0.33                     | 17.90    | 1.97 9.08 24.54 |
| pr124    | 8         | 1.83          | 0.48                     | 30.38    | 2.60 11.68 7.76 |
| bier127  | 8         | 1.78          | 0.42                     | 28.37    | 2.56 11.07 11.56 |
| ch130    | 8         | 1.78          | 0.42                     | 27.06    | 2.58 10.47 16.83 |
| pr136    | 8         | 2.35          | 0.55                     | 36.67    | 3.08 11.89 19.58 |
| gr137    | 8         | 1.63          | 0.46                     | 28.54    | 2.99 9.55 17.73 |
| pr144    | 8         | 8.51          | 1.28                     | 108.24   | 5.38 20.11 2.01 |
| ch150    | 8         | 2.26          | 0.61                     | 36.25    | 3.75 9.66 24.69 |
| kroA150  | 8         | 2.74          | 0.69                     | 42.51    | 3.90 10.89 18.29 |
| kroB150  | 8         | 2.96          | 0.73                     | 44.23    | 3.97 11.14 17.64 |
| pr152    | 8         | 7.77          | 1.27                     | 108.04   | 5.87 18.41 5.08 |
| Average for \( ng = 8 \) | | 4.37 | | 12.18 | 15.06 |
| gr120    | 12        | 4.74          | 0.82                     | 53.77    | 2.98 18.04 19.40 |
| pr124    | 12        | 38.77         | 15.66                    | 203.96   | 9.73 20.96 4.03 |
| bier127  | 12        | 8.40          | 1.46                     | 85.20    | 4.24 20.09 11.58 |
| ch130    | 12        | 7.60          | 1.14                     | 81.84    | 3.98 20.58 16.33 |
| pr136    | 12        | 11.40         | 1.80                     | 120.52   | 5.44 22.17 13.50 |
| gr137    | 12        | 7.87          | 1.41                     | 95.40    | 4.93 19.33 13.96 |
| pr144    | 12        | 174.41        | 33.38                    | 555.34   | 19.80 28.05 1.62 |
| ch150    | 12        | 8.55          | 1.27                     | 102.46   | 5.09 20.11 22.03 |
| kroA150  | 12        | 12.12         | 1.62                     | 129.82   | 5.82 22.32 16.56 |
| kroB150  | 12        | 14.06         | 2.04                     | 140.31   | 6.37 22.02 15.14 |
| pr152    | 12        | 212.33        | 29.70                    | 731.47   | 23.54 31.08 2.28 |
| Average for \( ng = 12 \) | | 6.01 | | 22.25 | 12.40 |
| gr120    | 16        | 28.68         | 5.21                     | 158.58   | 6.84 23.18 12.30 |
| pr124    | 16        | -             | -                        | -        | -    -    -    |
| bier127  | 16        | 57.66         | 8.99                     | 256.19   | 9.41 27.22 9.46 |
| ch130    | 16        | 35.69         | 4.35                     | 210.17   | 7.62 27.58 13.70 |
| pr136    | 16        | 65.27         | 8.96                     | 334.53   | 13.12 25.51 10.41 |
| gr137    | 16        | 48.09         | 8.84                     | 286.45   | 11.56 24.78 10.08 |
| pr144    | 16        | -             | -                        | -        | -    -    -    |
| ch150    | 16        | 40.89         | 5.12                     | 279.85   | 9.47 29.55 16.99 |
| kroA150  | 16        | 66.00         | 6.69                     | 360.41   | 11.21 32.16 13.62 |
| kroB150  | 16        | 89.06         | 11.86                    | 409.30   | 13.98 29.28 12.40 |
| pr152    | 16        | -             | -                        | -        | -    -    -    |
| Average for \( ng = 16 \) | | 7.28 | | 27.41 | 12.37 |

6. Conclusions

This paper dealt with the Minimum Latency Problem (MLP), a variant of the Traveling Salesman Problem (TSP) where the objective is to minimize the sum of waiting times of customers.
A branch-and-price (BP) algorithm over a set partitioning formulation was introduced, where columns are $ng$-paths. As implemented by Roberti and Mingozzi (2014), our algorithm is based on dynamically defined $ng$-memories. The proposed BP benefits from well-known elements of efficient exact methods for routing problems, such as dual stabilization, reduced cost fixing, route enumeration and strong branching.

Although those elements are very important for the good performance of BP, the main sources of efficiency of our method are the new features for the $ng$-path relaxation. For example, the new dominance rules typically speed up the labeling algorithm by factors between 4 and 8. Furthermore, the proposed generalized definition of $ng$-sets in terms of arcs opened the way for less harmful augmentations of $ng$-memories, in contrast to the vertex-based augmentations introduced by Roberti and Mingozzi (2014). Nevertheless, our experiments showed that the best strategy for augmenting $ng$-memories is instance-dependent: some of them are better solved with arc-based augmentations, others with vertex-based augmentations. For example, pr124 could not be solved in 2 days using only arc-based augmentations, while pr136, pr144 and p152 were solved for the first time mainly because of them. Hence, for the sake of robustness, the solution of a node in our BP starts with vertex-based augmentations, switching to arc-based ones if the computational time of the labeling algorithm is too large. The proposed $ng$-memories reduction algorithm is crucial in this transition, reverting previous augmentations that are not needed to attain the current bound. We should mention that we did try to use this reduction of $ng$-memory more frequently, but it is time-consuming for large instances and affects the convergence of Algorithm 7. Still, we believe that the combination of augmentations and reductions of $ng$-memories deserves further investigation.

All the 9 instances not solved by Roberti and Mingozzi (2014) were solved. Also, BP could solve the larger instances u159, si175, br180 and rat195, but could not solve d198, kroA200 and kroB200. In general, MLP instances with about 150 vertices (or more) are still very challenging.

**Acknowledgments**

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d’Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the “Programme d’Investissements d’Avenir” (see [www.plafrim.fr/en/home](http://www.plafrim.fr/en/home)). TB and EU visited Bordeaux with funding from project FAPERJ E26/110.075/2014 (International Cooperation FAPERJ/INRIA). We also thank Marcos Melo Silva and Anand Subramanian for computing upper bounds for the large instances tested only in this paper.
References


URL https://realopt.bordeaux.inria.fr/?page_id=2