Dantzig-Wolfe Reformulation for the Network Pricing Problem with Connected Toll Arcs

Samuel Deleplanque[a,b], Bernard Fortz[a,b], Martine Labbé[a,b], Alessia Violin[a]

[a] *Département d'Informatique*
Université Libre de Bruxelles
Brussels, Belgium
[b] *INOCS, INRIA Lille Nord-Europe, France*

Abstract

This paper considers a pricing problem on a network with connected toll arcs and proposes a Dantzig-Wolfe reformulation for it. First, the relaxation of this formulation is theoretically shown to be at least as good as the reference proposed in the literature. Then, we detail the particularities of the implementation of a branch-and-price algorithm for solving it such as a primal heuristic, the branching rules, the stabilization process, and an algorithm for fixing variables using Lagrangian duality. Finally, numerical results on two types of instances, real and pseudo-randomly generated, are reported, confirming numerically the main theoretical results.

Keywords: Combinatorial Optimization, Bilevel Programming, Pricing, Networks, Dantzig-Wolfe.

1. Introduction

Several real-world problems involve a hierarchical relationship between two decision levels, in areas like management (facility location, environmental regulation), economic planning (electric power pricing, oil production), or engineering (optimal design, structures and shape). In real-world sequential models, the choice of a decision-maker, i.e. the upper level, leads to some reaction within a particular market or social entity, which corresponds to the lower level of the problem (Colson et al. (2007)).

In mathematical programming a Bilevel Programming (BP) problem is a hierarchical optimization problem in which part of the constraints translates

the fact that some of the variables constitute an optimal solution to a second optimization problem. These problems were introduced by Bracken and McGill (1973) as mathematical programs with optimization problems in the constraints, whereas the terms bilevel and multilevel were later introduced by Candler and Norton (1977).

In this setting the first objective function and its proper constraints, which are not related to the second optimization problem, usually refer to the so-called leader or first level, whilst the second optimization problem (objective function and constraints) refers to the follower or second level. This terminology reflects the sequentiality of the problem: the follower chooses his/her optimal solution once the leader's choice is known, and the leader will therefore optimize his/her choice taking into account that the follower always reacts optimally to it.

From a computational point of view, bilevel problems are intrinsically difficult. Even the simple version of a (BP) where the objective functions and the constraints are linear has been shown to be $\mathcal{NP}$-hard by Jeroslow (1985). Furthermore, Hansen et al. (1992) prove the strong $\mathcal{NP}$-hardness of these problems. Vicente et al. (1994) strengthen these results and prove that merely checking strict local optimality and checking local optimality in linear (BP) are $\mathcal{NP}$-hard problems.

Due to the difficulty of (BP), solution methods and algorithms generally focus on particular cases where functions have convenient properties, such as linearity or convexity, in order to exploit their structure to develop efficient solution methods. References can be found in Vicente and Calamai (1994); Dempe (2002); Colson et al. (2005, 2007). Refer to Labbé and Violin (2013) for more detail on the bilevel programming and price setting problems.

This paper deals with a pricing problem. In such an optimization problem, the leader (first level) sets some taxes or prices for some activities, and the followers (second level) select activities from among taxed and untaxed ones to minimize operating costs. We assume that there are n_1 taxed and n_2 untaxed activities. By setting $T \in \mathbb{R}^{n_1}$ as the tax vector, $x \in \mathbb{R}^{n_1}$ and $y \in \mathbb{R}^{n_2}$ as the vectors associated with taxed and untaxed activities respectively, f and g as the objective functions of the leader and the follower respectively, and $\Pi \subset \mathbb{R}^{n_1+n_2}$ as the feasible solution set, the pricing problem can be formulated as:

$$\max_{T} \quad f(T, x, y), \tag{1a}$$

$$\text{s.t.} \quad (x, y) \in \arg\min_{x,y} g(T, x, y), \tag{1b}$$

$$(x, y) \in \Pi. \tag{1c}$$

The problem we consider in this paper is the Network Pricing Problem (NPP), with an authority which owns a subset of arcs and imposes tolls on them, and users who travel on the network as in Labbé et al. (1998). The authority is the leader who wants to maximize his/her revenue, and network users are the followers who want to minimize their costs, and so will always travel on their minimum cost path. The revenue of the leader is yielded by the tolls crossed by the followers.

One interesting situation with a polynomial number of paths is the highway system: it is characterized by a network whose connected toll arcs constitute a single path. This variant of the NPP has been considered in Heilporn et al. (2010, 2011). If we make the assumption that users who have left the highway do not re-enter it, paths considered for toll can be uniquely determined by their entry and exit nodes. In consequence, for n nodes in the highway, the number of total paths will be $n(n-1)$. Because of the completeness of the toll subgraph, this problem is also called the clique pricing problem.

In this paper, we present a method to solve medium size instances of the Network Pricing Problem with Connected Toll Arcs. It is organized as follows. Section 2 is devoted to two formulations: a bilevel and a single level, both from the literature. In Section 3, we propose a Dantzig-Wolfe reformulation (presented in Fortz et al. (2013)), the Subproblem and the Master Problem are given. Then, a theoretical comparison between the formulations is made. Section 4 describes the implementation of the Dantzig-Wolfe reformulation. It gives a way to develop a Branch-and-Price with a focus on a primal heuristic, a stabilization algorithm, branching strategies, and fixing variables rules using Lagrangian duality. Experiments in Section 5 consider two types of instances: one from real data (from italian highways) and another generated pseudo-randomly.

2. Problem definition and formulations

The transportation network is defined as a set of nodes linked by a complete set of directed arcs. Each node represents an entry or an exit node of the highway and each arc the subpath of the highway linking the arc extremities. We define $\mathcal{N}$ as the set of nodes i, $\mathcal{A}$ as the set of toll arcs (i.e. arcs owned by the leader on which he/she can impose tolls). We denote as a the generic arc.

The set $\mathcal{K}$ represents the commodities k, which are groups of network users traveling from an origin to a destination. For each commodity $k \in \mathcal{K}$, let η^k be its demand and o^k and d^k be its origin and destination respectively. Moreover, each toll path $a \in \mathcal{A}$ has a fixed cost c_a^k and the shortest toll free path has another fixed cost c_{od}^k. The fixed cost c_a^k is given by the sum of the costs for traveling from o^k to the origin of arc a, along arc a and from the extremity of arc a to d^k.

The leader wants to set a toll T_a on each toll arc $a \in \mathcal{A}$, such that his/her total revenue is maximum, and followers seek their minimum total cost (fixed cost plus toll) path on the network.

We assume that for each commodity there exists a toll free path between its origin and destination, i.e. a path which does not pass through any of the arcs owned by the authority, as otherwise the authority could impose an infinite toll on his/her arcs to obtain infinite revenues.

Figure 1 depicts the graph involved in the so-called Complete Toll NPP, introduced in Heilporn et al. (2010): toll free arcs are inserted between origin and destination nodes, as well as from the origin and destination nodes to the highway, representing minimum cost toll free paths between the corresponding nodes.

Each pair of entry and exit nodes of the highway is linked by a toll subpath. Each toll subpath can be represented by a single artificial toll arc (dotted arcs in Figure 1). In this a case additivity conditions are not considered, meaning that the toll of a path might not be equal to the sum of tolls on subpaths composing it.

There are also some cases where the follower has multiple optimal solutions for a given toll vector. Here we consider that in such cases the commodities take the choice which is most profitable for the leader. This assumption is not restrictive, because the leader could modify some tolls of his/her most profitable solution by a small value, making that solution the only optimal one for the follower.

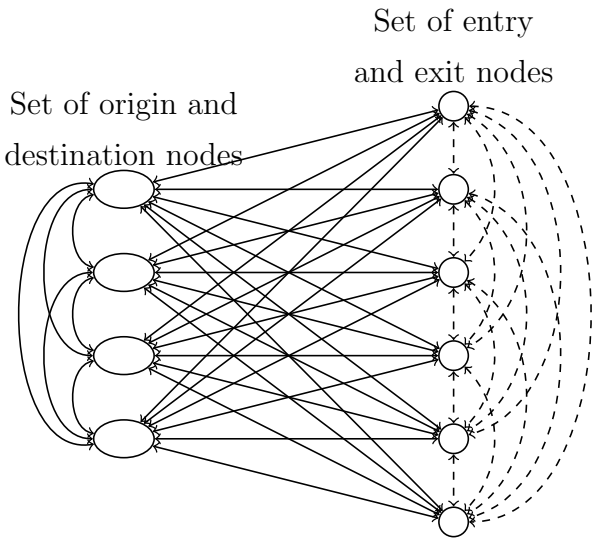

Figure 1: Complete toll NPP (from Heilporn et al., 2010)

2.1. Bilevel formulation

We consider flow variables x_a^k on toll arcs, and y_{od}^k on toll free arcs taking value 1 if commodity k uses arc a and 0 otherwise. The Network Pricing Problem (NPP) can be expressed as follows:

$$\max_{T \geq 0} \quad \sum_{k \in \mathcal{K}} \eta^k \sum_{a \in \mathcal{A}} T_a x_a^k, \tag{2a}$$

$$\text{s.t. } (x, y) \in \arg\min_{x,y} \quad \sum_{k \in \mathcal{K}} \left(\sum_{a \in \mathcal{A}} (c_a^k + T_a) x_a^k + c_{od}^k y_{od}^k \right), \tag{2b}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}} x_a^k + y_{od}^k = 1 \qquad \forall k \in \mathcal{K}, \tag{2c}$$

$$x_a^k, y_{od}^k \in \{0, 1\} \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}. \tag{2d}$$

where constraints (2c) allow one path choice for each commodity.

The multicommodity NPP has been proved to be strongly $\mathcal{NP}$-hard by Heilporn et al. (2010), whether toll arcs are single or bidirectional, based on a reduction from the 3-SAT problem. The NPP with only one toll subpath (i.e. single toll arc) is polynomial.

2.2. One level non-linear and MILP formulations

NPP can be reformulated as a single level optimization problem. First, the follower's objective function can be separated per commodity, and the lower level optimization problem can be replaced by constraints stating explicitly that the used path is the shortest one ((3b), (3c)). Moreover, variables y_{od}^k, associated with toll free paths, can be eliminated using constraints (2c).

We introduce a new set of variables $p_a^k = x_a^k T_a$, yielding to only one non linear set of constraints (3e) in the following model. We note (HPNL) the non-linear and single level formulation. It can be written as follows:

(HPNL)

$$\max_{T,x,p} \quad \sum_{a\in\mathcal{A}}\sum_{k\in\mathcal{K}} \eta^k p_a^k, \tag{3a}$$

$$\text{s.t.} \quad \sum_{a\in\mathcal{A}} \left[(c_a^k - c_{od}^k)x_a^k + p_a^k\right] - T_b \leq c_b^k - c_{od}^k \qquad \forall k \in \mathcal{K}, \forall b \in \mathcal{A}, \tag{3b}$$

$$\sum_{a\in\mathcal{A}} \left[(c_a^k - c_{od}^k)x_a^k + p_a^k\right] \leq 0 \qquad \forall k \in \mathcal{K}, \tag{3c}$$

$$\sum_{a\in\mathcal{A}} x_a^k \leq 1 \qquad \forall k \in \mathcal{K}, \tag{3d}$$

$$p_a^k = T_a x_a^k \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{3e}$$

$$x_a^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{3f}$$

$$T_a \geq 0 \qquad \forall a \in \mathcal{A}. \tag{3g}$$

This (HPNL) model can be linearized eliminating the non linear set of constraints (3e), introducing the following set of constraints, $\forall k \in \mathcal{K}$ and $\forall a \in \mathcal{A}$:

$$p_a^k \leq M_a^k x_a^k \tag{4a}$$

$$T_a - p_a^k \leq N_a(1 - x_a^k) \tag{4b}$$

$$p_a^k \leq T_a \tag{4c}$$

The mixed integer linear model that we note (HPL) is therefore (Heilporn et al., 2010):

(HPL)

$$\max_{T,x,p} \quad \sum_{a\in\mathcal{A}}\sum_{k\in\mathcal{K}} \eta^k p_a^k, \tag{5a}$$

$$\text{s.t.} \quad \sum_{a\in\mathcal{A}} \left[(c_a^k - c_{od}^k)x_a^k + p_a^k\right] - T_b \leq c_b^k - c_{od}^k \qquad \forall k \in \mathcal{K}, \forall b \in \mathcal{A}, \tag{5b}$$

$$\sum_{a\in\mathcal{A}} \left[(c_a^k - c_{od}^k)x_a^k + p_a^k\right] \leq 0 \qquad \forall k \in \mathcal{K}, \tag{5c}$$

$$\sum_{a\in\mathcal{A}} x_a^k \leq 1 \qquad \forall k \in \mathcal{K}, \tag{5d}$$

$$p_a^k \leq M_a^k x_a^k \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{5e}$$

$$T_a - p_a^k \leq N_a(1 - x_a^k) \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{5f}$$

$$p_a^k \leq T_a \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{5g}$$

$$p_a^k \geq 0 \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}, \tag{5h}$$

$$x_a^k \in \{0, 1\} \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}. \tag{5i}$$

Constants M_a^k represent upper bounds on p_a^k variables, i.e. the highest value that commodity k is willing to pay for path a. Therefore, a valid value is $M_a^k = \max\{0, c_{od}^k - c_a^k\}$, $\forall k \in \mathcal{K}$ and $\forall a \in \mathcal{A}$. Constants N_a represent upper bounds on the cost of a path for all commodities, and a valid value is $N_a = \max_{k\in\mathcal{K}:a\in\mathcal{A}} M_a^k$, $\forall a \in \mathcal{A}$ (Heilporn et al., 2010).

3. A Dantzig-Wolfe reformulation

We note (HPDW) a Dantzig-Wolfe reformulation of the NPP with connected toll arcs starting from the single level non-linear model (HPNL). This reformulation (HPDW) is defined below by its Master Problem and its Subproblem (*Pricing*). Then, we study (HPDW) theoretically and compare its linear relaxation to that of (HPL).

3.1. Master Problem

The Master Problem (MP) is composed of the objective function (3a) and constraints (3b), (3d) and (3f). Our sets $\mathcal{X}_a$, defining the feasible region of the subproblem associated to path $a \in \mathcal{A}$, are represented by points satisfying

constraints (3e), (3f) and (3g), which are separable for each path $a \in \mathcal{A}$. Moreover we add to the subproblem the following set of constraints:

$$p_a^k \leq M_a^k x_a^k \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}. \tag{6}$$

It is straightforward to verify that these constraints are valid, as they derive from the linearisation of constraints (3e). Moreover, if we aggregate them summing over all paths $a \in \mathcal{A}$, we obtain constraints (3c), such that by including constraints (6) in the subproblem we verify constraints (3c) and we can omit them in the (MP).

We represent a feasible solution of the subproblem by a point $(x_a^k, T_a, p_a^k)^j$, for $j \in \mathcal{J}$ where $\mathcal{J}$ is the space of feasible points for each a, k, and we associate to it a variable $\lambda_a^j \geq 0$. Note that sets $\mathcal{X}_a$ are not discrete, as we have binary variables x_a^k together with continuous variables T_a and p_a^k. Let us denote $(x_a^{k,j}, T_a^j, p_a^{k,j})$ as the j^{th} point of a set $\mathcal{X}_a$. For each $a \in \mathcal{A}$ we have:

$$\begin{pmatrix} x_a^k \\ T_a \\ p_a^k \end{pmatrix} = \sum_{j \in \mathcal{J}} \lambda_a^j \begin{pmatrix} x_a^{k,j} \\ T_a^j \\ p_a^{k,j} \end{pmatrix} = \lambda_a^1 \begin{pmatrix} x_a^{k,1} \\ T_a^1 \\ p_a^{k,1} \end{pmatrix} + \lambda_a^2 \begin{pmatrix} x_a^{k,2} \\ T_a^2 \\ p_a^{k,2} \end{pmatrix} + ... + \lambda_a^{|\mathcal{J}|} \begin{pmatrix} x_a^{k,|\mathcal{J}|} \\ T_a^{|\mathcal{J}|} \\ p_a^{k,|\mathcal{J}|} \end{pmatrix}. \tag{7}$$

We also impose $\sum_{j \in \mathcal{J}} \lambda_a^j = 1$ for each $a \in \mathcal{A}$ (convexity constraint). Note that as sets $\mathcal{X}_a$ are not discrete λ_a^j variables do not need to be binary. However, the integrality of the original variables x_a^k is guaranteed by keeping constraints (3f) also in the (MP).

We therefore express the (MP) using the convex combination of the feasible points of sets $\mathcal{X}_a$:

(MP)

$$\max_{\lambda} \quad \sum_{a\in\mathcal{A}}\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}} \lambda_a^j \eta^k p_a^{k,j}, \tag{8a}$$

$$\text{s.t.} \quad \sum_{a\in\mathcal{A}}\sum_{j\in\mathcal{J}} \left[(c_a^k - c_{od}^k)\lambda_a^j x_a^{k,j} + \lambda_a^j p_a^{k,j}\right] - \sum_{j\in\mathcal{J}} \lambda_b^j T_b^j \leq c_b^k - c_{od}^k \quad \forall k \in \mathcal{K}, \forall b \in \mathcal{A}, \tag{8b}$$

$$\sum_{a\in\mathcal{A}}\sum_{j\in\mathcal{J}} \lambda_a^j x_a^{k,j} \leq 1 \qquad \forall k \in \mathcal{K}, \tag{8c}$$

$$\sum_{j\in\mathcal{J}} \lambda_a^j = 1 \qquad \forall a \in \mathcal{A}, \tag{8d}$$

$$\sum_{j\in\mathcal{J}} \lambda_a^j x_a^{k,j} \in \{0,1\} \qquad \forall a \in \mathcal{A}, \forall k \in \mathcal{K}, \tag{8e}$$

$$\lambda_a^j \geq 0 \qquad \forall j \in \mathcal{J}, \forall a \in \mathcal{A}. \tag{8f}$$

The number of variables λ_a^j is huge but (MP) is linear. As we want to solve the (MP) within a Branch-and-Price framework, first of all we solve the linear relaxation of (MP), eliminating constraints (8e).

For a given node, the column generation algorithm does not solve the linear relaxation of (MP) but the linear relaxation of the Restricted Master Problem (RMP). The algorithm iteratively solves (RMP), generating a subset of variables $\mathcal{J}'$, $\mathcal{J}' \in \mathcal{J}$, until the Subproblem proves that the algorithm has reached the optimality of the linear relaxation.

3.2. Subproblem

The pricing Subproblem (SP) is used to verify whether the linear relaxation of the master problem (by the constraint (8e)) restricted to a subset of variables is optimal, i.e. if all possible variables have a non positive reduced price, and, if not, to search for new variables (columns) to add to improve the LP-relaxation (MP) solution.

Solutions of (SP) must verify the constraints of the original problem (HPNL) that were not included in (MP): (3c), (3e) and (3g), plus constraints (3f) included in both (MP) and (SP). Constraints (3c) are satisfied by including in (SP) the following set of constraints where $M_a^k = \max\{0, c_{od}^k - c_a^k\}$:

$$p_a^k - M_a^k x_a^k \leq 0 \qquad \forall k \in \mathcal{K}, \forall a \in \mathcal{A}. \tag{9}$$

These constraints, combined with constraints (3e), reduce the domain of variables T_a and p_a^k: if $x_a^k = 0$ these constraints impose $p_a^k = 0$, whereas if $x_a^k = 1$ they become $T_a \leq M_a^k$, which is a valid upper bound on the toll if commodity k uses path a. Furthermore, they guarantee that in the subproblem there are no solutions with $x_a^k = 1$ if the corresponding $T_a > 0$ and $M_a^k = 0$.
We define the dual variables associated to constraints of (MP) as follows:

- variables $\delta_b^k \geq 0$ are associated to constraints (8b), one for each $b \in \mathcal{A}$ and $k \in \mathcal{K}$;
- variables $\gamma^k \geq 0$ are associated to constraints (8c), one for each $k \in \mathcal{K}$;
- variables $\mu_a \gtreqless 0$ are associated to constraints (8d), one for each $a \in \mathcal{A}$.

The reduced price of a variable λ_a^j corresponding to point $(x_a^{k,j}, T_a^j, p_a^{k,j})$ of the (MP) can be expressed as follows:

$$RP_a^j = \sum_{k \in \mathcal{K}} \eta^k p_a^{k,j} - \left\{ \sum_{k \in \mathcal{K}} \sum_{b \in \mathcal{A}} \delta_b^k \left[(c_a^k - c_{od}^k) x_a^{k,j} + p_a^{k,j} \right] - \sum_{k \in \mathcal{K}} \delta_a^k T_a^j + \sum_{k \in \mathcal{K}} x_a^{k,j} \gamma^k + \mu_a \right\}. \tag{10}$$

The subproblem (SP) is therefore:

$$\text{(SP)} \qquad \max_{a \in \mathcal{A}} \max_{j:(x_a^{k,j}, T_a^j, p_a^{k,j}) \in \mathcal{A}} RP_a^j. \tag{11a}$$

The reduced price RP_a^j is defined for each path a and we noticed that (SP) constraints are separable for each path a, meaning that we can solve a smaller subproblem for each path $a \in \mathcal{A}$, which can be formulated as follows:

(SP_a)

$$\max_{T,x,p} \quad \sum_{k\in\mathcal{K}} \eta^k p_a^k - \left[\sum_{k\in\mathcal{K}}\sum_{b\in\mathcal{A}} \delta_b^k \left(p_a^k - M_a^k x_a^k\right) - \sum_{k\in\mathcal{K}} \delta_a^k T_a + \sum_{k\in\mathcal{K}} x_a^k \gamma^k + \mu_a\right], \tag{12a}$$

$$\text{s.t} \quad p_a^k - M_a^k x_a^k \leq 0 \qquad \forall k \in \mathcal{K}, \tag{12b}$$

$$p_a^k = T_a x_a^k \qquad \forall k \in \mathcal{K}, \tag{12c}$$

$$x_a^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \tag{12d}$$

$$T_a \geq 0. \tag{12e}$$

Proposition 1. *The Subproblem (SP) used to solve (MP) (Equations (8a) to (8f)) is separable for each path $a \in \mathcal{A}$ and each $(SP)_a$ achieves its optimal value in $\{M_a^k : k \in \mathcal{K}\} \cup \{0\}$, so only $|\mathcal{K}| + 1$ values of T_a must be evaluated for solving (SP_a).*

Proof. This problem is non-linear due to (12c), but it is easy to solve, as we are solving a problem for a single path a. We notice that if the value of x_a^k is fixed for all $k \in \mathcal{K}$, then the objective function becomes linear in T_a, and the commodities path choice may change only at toll values $T_a \in \{M_a^k : k \in \mathcal{K}\} \cup \{0\}$.

Any other toll value can be increased without changing the commodities path choice and therefore increasing the objective function value. We then consider only these breakpoint values for T_a and we deduce the assignment values of commodities to toll paths x_a^k with the following procedure. We can rewrite the reduced price, indicating explicitly the coefficient of x_a^k variables (remember that $p_a^k = T_a x_a^k$):

$$RP_a = \sum_{k\in\mathcal{K}} x_a^k \left[T_a \left(\eta^k - \sum_{b\in\mathcal{A}} \delta_b^k\right) + \sum_{b\in\mathcal{A}} M_a^k \delta_b^k - \gamma^k\right] + \sum_{k\in\mathcal{K}} \delta_a^k T_a - \mu_a. \tag{13}$$

Denote as π_a^k the coefficient associated to x_a^k in the reduced price, $\forall a \in \mathcal{A}$ and $\forall k \in \mathcal{K}$:

$$\pi_a^k = T_a \left(\eta^k - \sum_{b \in \mathcal{A}} \delta_b^k \right) + \sum_{b \in \mathcal{A}} M_a^k \delta_b^k - \gamma^k. \tag{14}$$

For each $T_a \in \{M_a^k : k \in \mathcal{K}\} \cup \{0\}$, we calculate the value of π_a^k using Equation (14) and the assignment of commodities to paths that maximizes the reduced price is as follows:

$$x_a^k = \begin{cases} 1 & \text{if } M_a^k \geq T_a \text{ and } \pi_a^k > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

and then we calculate the corresponding reduced price:

$$RP_a(T_a) = \sum_{k \in \mathcal{K}} x_a^k \pi_a^k + \sum_{k \in \mathcal{K}} \delta_a^k T_a - \mu_a. \tag{16}$$

We choose the optimal value $T_a^* = \arg\max_{T_a}\{RP_a(T_a) : T_a \in \{M_a^k : k \in \mathcal{K}\} \cup \{0\}\}$ and the corresponding assignment values. Each $(\text{SP})_a$ is solved evaluating $|\mathcal{K}| + 1$ values, such that to solve (SP) we need to evaluate $O(|\mathcal{A}||\mathcal{K}|)$ values. □

For each path a the number of total columns is bounded by all the possible values for the toll, which is $O(|\mathcal{K}|)$, multiplied by all the possible assignment matrices of the commodities, which is $2^{|\mathcal{K}|}$. The total number of columns is therefore $O(|\mathcal{A}||\mathcal{K}|2^{|\mathcal{K}|})$ which is exponential as expected.

3.3. Comparison between the formulations

We compare the different formulations we have for the NPP with connected toll arcs, analyzing the optimal value of their linear relaxation compared to the optimal integer solution value. Consider the linear relaxation of (HPL) and (HPDW) and note them as (HPL-LR) and (HPDW-LR) respectively. We define as $F(*)$ the set of feasible solutions for a formulation $*$.

Proposition 2. *F(HPDW-LR) $\subseteq$ F(HPL-LR).*

Proof. Let us describe $F(\text{HPNL}) = \{x \in X : Ax \leq b, x \in \{0,1\}\}$, where X is the set of points satisfying constraints (3e), (3f) and (3g), $Ax \leq b$ corresponds to constraints (3b), (3c) and (3d), and $x \in \{0,1\}$ to constraints (3f).

It is obvious that $F(\text{HPL}) = F(\text{HPNL})$ and we defined (HPDW) such that $F(\text{HPDW}) = \{x \in conv(X) : Ax \leq b, x \in \{0,1\}\}$.

We conclude that $F(\text{HPDW-LR}) \subseteq F(\text{HPL-LR})$ (see Geoffrion, 1974). □

Therefore the linear relaxation of the Dantzig-Wolfe reformulation provides an upper bound for the optimal value of the integer problem as least as good as the linear relaxation of the linear model (HPL).

More generally, the linear relaxation of the Dantzig-Wolfe reformulation always finds an optimal integer solution for the one toll path case, as it intrinsically explores all the feasible integer solutions.

Proposition 3 (Integrality of (HPDW) for the one toll path case)**.** *If we consider a network with only one toll path, the optimal solution of (HPDW-LR) is integer.*

Proof. For this case $T \in \{M^k : k \in \mathcal{K}\} \cup \{0\}$. When solving (SP) we exactly explore these values for T and find corresponding integer assignment variables. Solving (MP) corresponds to choose a convex combination of these values, such that at optimality the toll and the corresponding assignment providing the bigger revenue will be chosen, giving an integer solution. □

4. A Branch and Price implementation

We propose several components for an implementation of our Dantzig-Wolfe decomposition. We propose here a Primal heuristic, branching strategies, and an algorithm fixing variables by using the Lagrangean multipliers. The first step of a column generation algorithm is the initialization of the Master Problem with a subset of columns (variables). We first describe the mechanism to generate these columns.

4.1. Columns initializing the Master Problem

A good initial set of columns may help the convergence of the column generation algorithm, even though it has been shown that sometimes excellent initial integer solutions may be detrimental to solve a linear program

with column generation. A discussion on this topic with references can be found in Lübbecke and Desrosiers (2005). For our highway problem it is quite straightforward to find an initial set of feasible columns for the (MP).

First of all we need one column per toll path, otherwise convexity constraints (8d) are not satisfied. Moreover, feasibility for the linear relaxation of the restricted (MP) all along the solution procedure with column generation is guaranteed if we have one column j for each path a where the coefficients $x_a^{k,j}$ are zero for each $k \in \mathcal{K}$ (meaning that all commodities are assigned to their toll free path).

Therefore we initialize our problem with one column j for each toll path $a \in \mathcal{A}$, with the toll equal to a big value (for instance N_a are valid values) and the corresponding $x_a^{k,j} = 0$, for each $k \in \mathcal{K}$. The p vector is equal to Tx, such that it is null. With this set of columns we can immediately find a feasible (integer) solution for (MP), whose value is zero. This value provides us also with a valid primal (lower) bound for the integer problem. However, this bound is not good.

As we would like to have a better initial (integer) solution, we could try to initialize the problem with more columns. Moreover, starting the column generation algorithm with more columns may help the convergence of the algorithm.

An intuitive heuristic to find an additional set of initial columns is the following one. We generate a column j for each path $a \in \mathcal{A}$ and we fix the toll T_a^j equal to the smallest M_a^k which is not zero. We then calculate the corresponding optimal assignment for commodities to paths ($x_a^{k,j}$ values) with these tolls, solving a shortest path problem for each commodity. The p vector is equal to Tx. This heuristic provides us with a feasible integer solution whose value is positive in most cases.

4.2. Primal Heuristic

We propose a heuristic algorithm for the NPP with connected toll arcs in order to find a good integer solution for each node of the tree. For our maximization problem, the integer solution obtained by this heuristic will give us a lower bound.

We can notice that, if all parameters are integer, variables T_a are integer as well, leading to a full integer formulation.

The heuristic starts from the fractional toll values $\widetilde{T}_a$ provided by solving the Dantzig-Wolfe reformulation with the column generation algorithm. Then, it constructs a feasible integer solution with the following algorithm:

Step 1: For each toll path $a \in \mathcal{A}$, set the toll equal to the maximum between the fractional value, rounded down, and the minimum M_a^k, for all $k \in \mathcal{K}$, as follows:

$$T_a = \max\{\lfloor \widetilde{T}_a \rfloor, \min_{k \in \mathcal{K}} M_a^k\} \qquad \forall a \in \mathcal{A}. \tag{17}$$

We round down the fractional values to have integer toll values to avoid numerical problems in the column generation algorithm, and we set the tolls equal to at least the smallest M_a^k as we noticed that it improves the solution.

Step 2: Find the assignment of commodities to toll paths (x_a^k values) when applying these tolls, searching for the shortest path for each commodity. Calculate the assignment matrix as follows, for all $k \in \mathcal{K}$:

- if $\min_{a \in \mathcal{A}}\{c_a^k + T_a\} \leq c_{od}^k$ then

$$x_{\overline{a}}^k = 1 \quad \text{where} \quad \overline{a} = \arg\min_{a \in \mathcal{A}}\{c_a^k + T_a\} \tag{18}$$

  and $x_a^k = 0$, for all $a \neq \overline{a}$,
- otherwise $x_a^k = 0$ for all $a \in \mathcal{A}$.

Step 3: Calculate the leader's revenue corresponding to this solution:

$$\mathcal{R}(H1) = \sum_{a \in \mathcal{A}} \sum_{k \in \mathcal{K}} \eta^k T_a x_a^k. \tag{19}$$

4.3. Stabilization

The first experiments showed us the need of a stabilization procedure to speed up the convergence of column generation when solving the relaxed (MP). This arises from observations that dual variables do not smoothly converge to their respective optima, but oscillate, without an observed regular pattern (Lübbecke and Desrosiers, 2005). Those oscillations can lead to "extreme columns" that have low chances of being in the optimal solution.

Furthermore, columns that will be in the optimal solution are often generated in the last iterations, when dual variables are close to their optimal

values (Merle et al., 1999). We selected an algorithm to limit this oscillation behavior.

A dual variable smoothing approach stabilization procedure to speed-up column generation has been introduced by Wentges (1997) and further developed by Pessoa et al. (2010). The basic idea is that at each iteration of the column generation procedure, we use a convex combination of these current values with the best known estimates.

Whenever we find better dual values, we update the best estimates. To guarantee the convergence to the correct optimal solution, the last iterations are done with the standard column generation without any stabilization.

Consider a scalar parameter Δ, such that $0 < \Delta \leq 1$. Let $\overline{\pi}$ be the current best known vector of dual multipliers, which is the vector providing the smallest (dual) upper bound (for a maximization problem) among all vectors that have been evaluated.

Let π_{MP} be the current solution of dual multipliers of the restricted (MP) on a certain iteration of column generation. Instead of using π_{MP} when solving the next subproblem, we solve it using a convex combination of these values, as follows:

$$\pi = \Delta\pi_{MP} + (1 - \Delta)\overline{\pi} \tag{20}$$

Moreover, if this vector of dual multipliers π improves the dual upper bound, we update the best know vector $\overline{\pi} = \pi$. Let Z_{MP} be the current primal lower bound (which is the current solution value of the (MP)), and $UB(\overline{\pi})$ the best upper bound found so far, calculated from the dual vector using equation:

$$UB = \sum_{k \in \mathcal{K}} \gamma^k + \sum_{a \in \mathcal{A}} \mu_a - \sum_{b \in \mathcal{A}} \sum_{k \in \mathcal{K}} M_b^k \delta_b^k + \sum_{a \in \mathcal{A}} \mathrm{RP}_a, \tag{21}$$

where RP_a is the reduced price associated to path a, calculated with equation (13). We have found the optimal solution for (MP) when:

$$gap = \frac{UB(\overline{\pi}) - Z_{MP}}{UB(\overline{\pi})} < \varepsilon, \tag{22}$$

where $\varepsilon \geq 0$ and sufficiently small. The convergence of this procedure is guaranteed by the two following lemmas.

Lemma 4.1 (from Wentges (1997))**.** *If the solution of the subproblem solved*

with vector π gives a column that already exists in the current (MP), then $UB(\pi) \leq UB(\overline{\pi}) - \Delta(UB(\overline{\pi}) - Z_{MP})$.

This lemma guarantees that the new dual solution is better than the old one of at least a factor of $\Delta(UB(\overline{\pi}) - Z_{MP})$.

Lemma 4.2 (from Pessoa et al. (2010))**.** *If the solution of the subproblem solved with vector π does not give a column with positive reduced price with respect to vector π_{MP} (called a "misprice"), then $UB(\pi) \leq UB(\overline{\pi}) - \Delta(UB(\overline{\pi}) - Z_{MP})$.*

Each misprice guarantees that the gap is reduced by at least a factor of $1/(1-\Delta)$, such that the total number of misprices is polynomially bounded. To avoid the need of estimating a proper value for ε that guarantees optimality due to misprices, if $gap \leq \overline{\varepsilon}$ (where $\overline{\varepsilon} > \varepsilon$) we set $\Delta = 1$ such that the last iterations are done with the standard column generation without stabilization. As at this stage we will be close to the optimal solution, dual variable values will be close to the optimal ones and the stabilization would not change much the convergence of column generation, which should be fast.

The previous lemmas guarantee convergence even when misprices occur, but the convergence can be slow. To avoid this inconvenient if an iteration of the algorithm with the stabilization does not improve the gap, we disable the stabilization ($\Delta = 1$) for the following iteration.

Performances of the stabilization approach highly depend on a good choice of the parameter Δ, and this can vary significantly for each instance. To avoid the need of tuning the algorithm Pessoa et al. (2013) proposed an auto-regulating strategy which is based on the gradient information.

Strategy 1: We update the parameter Δ accordingly to the current gap between the lower and upper bound of the (MP) (equation 22). We choose an initial value for Δ, $\Delta_{init} \in]0, 1]$ by tuning it within numerical tests. We choose also a value $\overline{\varepsilon}$ after which we stop the stabilization. Therefore our strategy is:

- $\Delta = \Delta_{init}$;
- if $gap < (1 - \Delta_{init})$ then $\Delta = 1 - gap$;
- if $gap < \overline{\varepsilon}$ then $\Delta = 1$.

As the gap is decreasing during the solving of (MP), in this strategy the parameter Δ is always increasing: at the beginning more weight is

given to the best known dual values whilst at the end more weight is given to the current ones. This strategy follows the intuition that at the beginning current dual values are more unstable and we need to stabilize more, whilst during the solving they converge to the optimum and less stabilization is needed.

Strategy 2: Following Pessoa et al. (2013) idea of an auto-adaptive scheme to choose Δ, we do not tune the parameter's choice a priori, but we let the column generation algorithm modify it during the solving. They propose to use the subgradient information to update the parameter: if the subgradient reveals that the best known dual vector is good, we give more weight to it, otherwise we give more weight to the current dual vector. The strategy works as follows:

- if $g(\pi)(\overline{\pi} - \pi_{MP}) > 0$ then $\Delta = f_{decr}(\Delta)$;
- else $\Delta = f_{incr}(\Delta)$;

where $g(\pi)$ is the subgradient calculated in π, and increasing and decreasing functions are as follows:

$$f_{incr}(\Delta) = \Delta + 0.1(1 - \Delta) \tag{23}$$

$$f_{decr}(\Delta) = \begin{cases} \Delta/1.1 & \text{if } \Delta \in [0.5, 1] \\ \max\{0.001, \Delta - 0.1(1 - \Delta)\} & \text{otherwise} \end{cases} \tag{24}$$

As for the other strategies we set $\Delta = 1$ if $gap < \overline{\varepsilon}$.

4.4. Branching Strategies

The concept of costs and estimations to generate the Branch-and-Bound tree has been introduced by Benichou et al. (1971) and pseudo-costs based branching strategies are implemented in many integer programming solvers.

Pseudo-costs are empirically derived quantities, suggesting the degradation on the objective function value that will result from fixing a variable value when branching. Pseudo-costs measure in a quantitative way the importance of the different integer variables and they forecast the deterioration of the functional value when forcing an integer variable from a non integer to an integer value.

There is an assumption that pseudo-costs do not vary greatly from one part to another of the tree and if necessary they can be revised during the course of the tree search. Their justification and use are based only on experimental results, which showed that the pseudo-costs tend to remain fairly constant, having the same order of magnitude along the tree except perhaps at a few nodes.

For each pair (a, k), $a \in \mathcal{A}$ and $k \in \mathcal{K}$, we maintain two quantities, a lower pseudo-cost and an upper pseudo-cost, which we denote as $P_a^{k,-}$ and $P_a^{k,+}$ respectively. The lower (resp. upper) pseudo-cost is an estimate of the per-unit change in the objective function from forcing the value of the corresponding original variable to be rounded down to zero (resp. up to one).

Let $\widetilde{x}_a^k$ be the current fractional solution value of the variable x_a^k and F the current objective function value. Define F^- and F^+ as the objective function values obtained after solving the linear relaxation of the problem with $x_a^k = 0$ and $x_a^k = 1$ respectively, as represented in Figure 2.

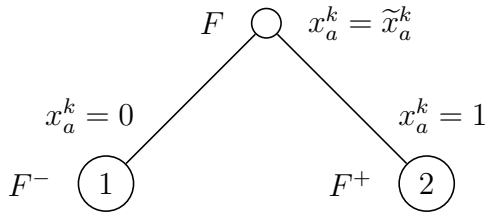

Figure 2: Branching on original variable x_a^k

Pseudo-costs of a pair (a, k) are therefore defined as follows:

$$P_a^{k,-} = \frac{F - F^-}{\widetilde{x}_a^k} \qquad \text{and} \qquad P_a^{k,+} = \frac{F - F^+}{1 - \widetilde{x}_a^k} \tag{25}$$

It may happen that in different parts of the tree we branch several times on the same pair: we keep an average of all pseudo-costs that are calculated during the solving for each pair.

Pseudo-costs have to be initialized at the root node. For each pair (a, k) we set both pseudo-costs as $P_a^{k,-} = P_a^{k,+} = 10\eta^k$, giving more weight to pairs with higher objective function parameter. Furthermore, during the tree search, we implemented two strategies to fix pseudo-costs for pairs (a, k) on which no branching was performed yet:

Pseudo-Cost calculation 1: keep the initial value calculated at the root node;

Pseudo-Cost calculation 2: use the average of all available pseudo-costs.

From pseudo-costs we deduce an estimation of the degradation in the objective function of both children resulting from branching on a pair (a, k) as follows:

$$D_a^{k,-} = P_a^{k,-}\widetilde{x}_a^k \qquad \text{and} \qquad D_a^{k,+} = P_a^{k,+}(1 - \widetilde{x}_a^k) \tag{26}$$

Therefore, we can measure the goodness for a potential branching pair (a, k) as a combination of the two estimated degradations. We choose the branching pair $(a, k)^*$ with one of the following strategies:

Branching pair choice strategy 1:

$$(a, k)^* = \arg \max_{a \in \mathcal{A}, k \in \mathcal{K}} \{\max\{0.00001, D_a^{k,-}\} \max\{0.00001, D_a^{k,+}\}\} \tag{27}$$

where we pay attention of not using null values for the estimated degradation to avoid numerical problems;

Branching pair choice strategy 2:

$$(a, k)^* = \arg \max_{a \in \mathcal{A}, k \in \mathcal{K}} \{\alpha \min\{D_a^{k,-}, D_a^{k,+}\} + (1 - \alpha) \max\{D_a^{k,-}, D_a^{k,+}\}\} \tag{28}$$

where $0 < \alpha \leq 1$ is a parameter to be set within numerical tests. If we choose $\alpha = 1$ we branch on the variable maximising the minimum degradation on the subtree, meaning that we try to quickly get nodes with optimal objective value smaller than the current best integer solution such that we can cut them off and reduce the size of the tree.

Furthermore, we define "quasi-integer" variables, which have almost integer current fractional value, such that they have good chances to take integer values when forcing other variables to be integer, without the need to branch on them. These are variables for which $0 < \widetilde{x}_a^k \leq \varsigma$ or $1 - \varsigma \leq \widetilde{x}_a^k < 1$, with $0 < \varsigma < 0.5$ (typically $0.1 \leq \varsigma \leq 0.2$).

When choosing the branching pair, we first select it among corresponding non quasi-integer variables (for which $\varsigma \leq \widetilde{x}_a^k \leq 1 - \varsigma$), and if there are none,

among corresponding quasi-integer variables. This idea is in line with the most fractional variable branching rule.

Finally, pseudo-costs allow us to calculate an estimate for each waiting node, such that the node with best estimate is chosen as the following one to be solved, determining the nodes sequence in the tree exploration. When branching on a pair (a, k) we denote 1 as the node obtained adding the branching decision $x_a^k = 0$ and 2 as the node obtained adding the branching decision $x_a^k = 1$, as represented in Figure 2. We propose two ways to set the estimate of nodes 1 and 2:

Estimation strategy 1: use the functional value of the best integer solution which can be expected at the descendant of a node:

$$E_{1,2} = F - \sum_{a \in \mathcal{A}, k \in \mathcal{K}} \min\{D_a^{k,-}, D_a^{k,+}\}; \tag{29}$$

Estimation strategy 2: use the functional value of the best expected (fractional) optimal solution:

$$E_1 = F - D_a^{k,-} \qquad \text{and} \qquad E_2 = F - D_a^{k,+} \tag{30}$$

There is also one additional parameter that we can set when creating children after branching, called priority of the node: in the tree exploration strategy, children are chosen before siblings and before leaves by priority if the estimate is good enough. This priority parameter gives preference between the two children of a node. We propose two strategies:

Node's priority strategy 1: use the default strategy, i.e. both priorities set to one;

Node's priority strategy 2: give more priority to the child with the smaller expected degradation, setting the priority to:

$$Pr_1 = -D_a^{k,-} \qquad \text{and} \qquad Pr_2 = -D_a^{k,+} \tag{31}$$

4.5. Fixing routing variables with the Lagrangian relaxation

The idea is to fix some $x_a^k, \forall a \in A, \forall k \in K$, variables as with the branching process. At the end of the Column Generation algorithm in some node, and

once we do not have any column to add to the Master problem, we take the last dual solutions obtained from the resolution of the Master Problem. At this point, the subproblem constituted by those dual solutions did not give us any positive reduced cost.

The dual solutions are in fact the Lagrangean multipliers that we need in this algorithm to fix the x_a^k boolean variables. We note $\delta_b^k, \forall b \in A, \forall \in K$, and $\gamma^k, \forall k \in K$, the Lagrangian multipliers related to the Master Problem's constraints (8b) and (8c) respectively. We denote by $\underline{Z}$ the current best feasible solution to (HPDW). $\underline{Z}$ is then a lower bound for our problem. We note $(\bar{x}_a^k, \bar{p}_a^k, \bar{T}_a)$ the last solution obtained from the resolution of the subproblem.

We note Z_{LP} the Lagrangian relaxation obtained by dualizing (8b) and (8c) with the optimal Lagrangian multipliers. It is at the same time the objective function value of the optimal solution of the LP relaxation of the Master Problem. Z_{LP} is given by the function (32a) followed by the remaining constraints (32b), (32c) and (32d).

$$\begin{aligned} Z_{LP} = Max & \sum_k \eta^k \sum_a \bar{p}_a^k \\ - \sum_k \delta_b^k [\sum_a [c_a^k - c_{od}^k] \bar{x}_a^k + p_a^k - \bar{T}_b - c_b^k + c_{od}^k] & \qquad (32a) \\ - \sum_k \gamma^k (\sum_a \bar{x}_a^k - 1) & \end{aligned}$$

s.t.

$$\bar{p}_a^k = \bar{T}_a \bar{x}_a^k, \quad \forall a \in A, \forall k \in K \qquad (32b)$$

$$\bar{p}_a^k \leq [c_{od}^k - c_a^k] \bar{x}_a^k \quad \forall a \in A, \forall k \in K \qquad (32c)$$

$$\bar{x}_a^k \in \{0, 1\} \quad \forall a \in A, \forall k \in K \qquad (32d)$$

We can deduce two boolean expressions in order to fix $x_a^k, \forall a \in A, \forall k \in K$. We can fix $x_a^k = 0$ if the boolean expression (33) is True. Otherwise, we also check if we can fix $x_a^k = 1$ if the boolean expression (34) is True.

$$(\bar{x}_a^k = 0) \wedge (Z_{LP} + \delta_b^k [c_{od}^k - c_a^k] - \gamma^k \leq \underline{Z}) \qquad (33)$$

$$(\bar{x}_a^k = 1) \wedge (Z_{LP} - \delta_b^k [c_{od}^k - c_a^k] + \gamma^k \leq \underline{Z}) \qquad (34)$$

5. Computational Experiments

The Dantzig-Wolfe reformulation (HPDW) has been experimentally compared with the single level formulation (HPL) on two types of instances defined below. We compare the linear relaxations of the two formulations. Finally, we present the results of the Branch-and-Price and Branch-and-Bound implementations for the (HPDW) and (HPL), respectively. The Branch-and-Price algorithm considered in these experiments is based on the description in the previous section.

For all the experiments, the computer used for these tests has an i7-4790 CPU clocked at 4.00GHz with 32Go of RAM. Each implementation has a maximum of 18,000 seconds (5 hours) to solve each individual instance.

5.1. Instances

A1 Instances. The experiments have been made on two different types of instances: the A1 instances and the complete graph instances. The former have been created in order to test our algorithms on realistic instances with an underlying network. The name "A1" corresponds to the italian "A1" highway which connects Milano to Napoli going from north to south of the country.

To construct the toll paths network we considered the principal entry and exit nodes of the highway, which are represented by the following fourteen cities: Milano, Lodi, Piacenza, Parma, Reggio Emilia, Modena, Bologna, Firenze, Arezzo, Orvieto, Roma, Frosinone, Caserta and Napoli. The real length of the highway's arcs is publicly available on-line (A1.Instances (2014 (Accessed)). We considered highway's subnetworks with 7, 11 and 14 subsequent entry and exit nodes, which correspond to 21, 55 and 91 toll paths respectively, as n nodes provide $n(n-1)/2$ pairs of entry/exit nodes (as all parameters are symmetrical we consider only the highway network in one direction).

Commodities are chosen between pairs of cities in a set composed by the cities on the highway nodes plus eight additional cities distributed in the surroundings of the highway (Brescia, Verona, Genova, Livorno, Perugia, L'Aquila, Latina and Salerno), for a total of 22 cities. We construct instances with 7, 11 and 14 cities randomly chosen in the set, corresponding to 21, 55 and 91 commodities respectively (a commodity is a pair of origin/destination cities).

The demand dem_{C1toC2} of each commodity, from a city $C1$ to a city $C2$, is calculated with their population pop_{C1} and pop_{C2}, respectively, and with their separating distance $dist_{C1,C2}$. The calculation uses the following simple gravitational model (Rosenberg, 2016):

$$dem_{C1toC2} = \frac{pop_{C1} * pop_{C2}}{dist^2_{C1,C2}} \tag{35}$$

where the population numbers have been obtained from publicly available on-line data and the distance between cities from *Google Maps*[1].

Finally, all distances on non highway arcs have been multiplied by a random value between 1.5 and 1.7, to take into account the speed difference when travelling on the highway or not (if we consider 120km/h as the average speed on the highway and 75 km/h on other routes we obtain a factor of 1.6).

Complete graph instances. The second set of instances have been created by the C++ standard random number generator. These "complete graph instances" have 20, 56 and 90 commodities and toll paths. They do not correspond to any underlying network, in the sense that we generate directly costs and demand data for each toll path and commodity.

Fixed costs on toll paths are randomly set between 1 and 20, and demand for commodities is randomly selected between 1 and 10. Complete network means that toll free path costs are randomly generated between 20 and 30 (and so are always bigger than fixed costs of toll paths), such that all commodities could use all toll paths.

5.2. (HPL) vs (HPDW) linear relaxations

We assess experimentally the linear relaxation of (HPDW), noted (HPDW-LR), by comparing with (HPL-LR). In Table 1 we report numerical results concerning the linear relaxation of (HPL) and (HPDW) formulations. We report the gap between the optimal integer value and the linear relaxation optimal value. Moreover, the time to solve the linear relaxation is given in seconds.

As expected from theoretical results, the gap of (HPDW-LR) is always smaller than the gap of (HPL-LR), with a greater difference between them when the number of toll paths is smaller or the number of commodities is

[1] www.google.com/maps

A1		21 commodities			55 commodities			91 commodities		
(sets of 10 inst. for each ($\|\mathcal{K}\|$,$\|\mathcal{A}\|$))		21 a	55 a	91 a	21 a	55 a	91 a	21 a	55 a	91 a
(-L)	Gap	5.30%	2.78%	1.41%	6.36%	3.57%	1.74%	6.26%	4.34%	3.12%
	Time	0.13	0.48	1.21	0.50	2.36	5.70	1.34	5.83	14.96
(-DW)	Gap	3.80%	2.30%	1.35%	3.14%	2.52%	1.25%	2.41%	2.97%	2.18%
	Time	0.07	0.22	0.46	0.48	1.69	3.85	3.14	8.63	16.45
Complete graph		20 commodities			56 commodities			90 commodities		
(sets of 10 inst. for each ($\|\mathcal{K}\|$,$\|\mathcal{A}\|$))		20 a	56 a	90 a	20 a	56 a	90 a	20 a	56 a	90 a
(-L)	Gap	6.19%	3.71%	3.15%	12.82%	11.18%	11.51%	15.33%	16.88%	17.47%
	Time	0.048	0.282	0.767	0.184	1.514	4.592	0.448	4.849	6.084
(-DW)	Gap	4.75%	3.52%	2.98%	7.72%	9.57%	10.68%	7.94%	13.94%	15.98%
	Time	0.199	0.726	1.57	4.13	12.7	22.3	25.84	62.26	108.99
	Time	5.51	7.79	11.39	43.12	47.04	56.83	198.38	221.28	260.09

Table 1: Numerical results on linear relaxation for (HPL) and (HPDW)

bigger. In particular, for instances with 90 commodities and 20 toll paths, (HPDW-LR) is able to halve the gap of (HPL-LR).

For the complete graph instances, (HPL-LR) has a smaller solution time than (HPDW-LR), with a large difference for big instances. However, for A1 instances (HPDW-LR) is competitive with respect to (HPL-LR), and for instances with 21 and 55 commodities it even outperforms it.

5.3. (HPDW) & Branch-and-Price vs (HPL) & Branch-and-Bound

We now compare the implementation of (HPDW) with the implementation of (HPL). The former consists of a Branch-and-Price using the SCIP 3.2's API tuned with the software `irace`, and the latter is a Branch-and-Bound using the same version of SCIP. Both solve the A1 instances and the Complete Graph instances.

Automatic configuration using `irace`. The implementation of (HPDW) gives us many parameters to be chosen such that manually testing all the possible configurations would require a big effort. Therefore, to determine its best configuration, we use `irace`, which is a tool for the automatic configuration of algorithms developed by López-Ibánez et al. (2011).

`irace` determined a set of good parameters for our (HPDW) implementation associated to the two types of instances. We noted these configurations $Tuned_{A1}$ and $Tuned_{CG}$ for the A1 instances and the complete graph instances, respectively. We compare these configurations with the "default" configuration named Try.

The table 2 reports a brief overview of the strategies for the stabilization process and the branching rules for the default configuration and the configurations selected by `irace`. Refer to Sections 4.3 and 4.4 for details.

	Try	$Tuned_{A1}$	$Tuned_{CG}$
Stabilization Strategy	Pessoa et al. (2013)	1-gap	Pessoa et al. (2013)
BS: Pseudo-Cost Calc.	Keep initial value	Keep initial value	Average value
BS: Branching Pair Rules	Product	Weighted sum	Product
BS: Node's Estimated Val.	Best integer sol.	Best fractional sol.	Best integer sol.
BS: Node's Priority	Smaller degradation	Smaller degradation	Balanced priorities

Table 2: Main strategies selected by `irace` for each type of instances

In Figures 3 and 4 we report the performance profile graphs where the y-axis represents the number of instances solved within the corresponding time (in seconds in logarithmic scale) in the x-axis, such that the higher the curve the better the configuration is performing.

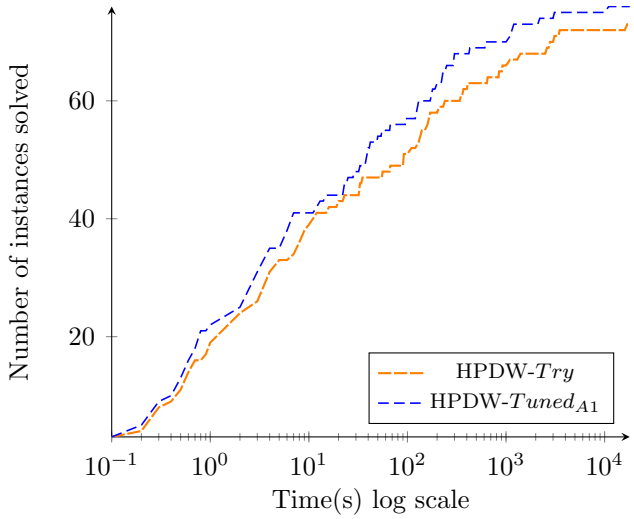

Figure 3: Performance profile graph on medium size A1 instances with (HPDW), comparing Try and $Tuned_{A1}$ configurations of SCIP

For the A1 instances, we can see that the two configurations give similar results. $tuned_{A1}$ is just slightly better for this type of instance. However, for the complete graph instances, we clearly see that the best configurations found by the tuning with `irace` outperform the configuration "Try".

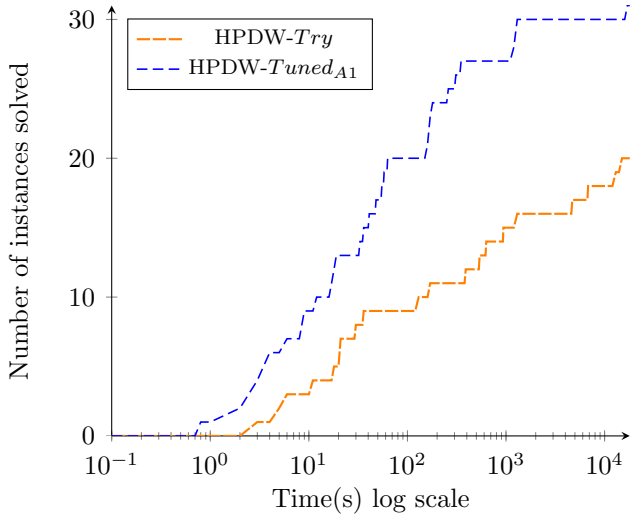

Figure 4: Performance profile graph on solving medium size complete graph instances with (HPDW), comparing Try and $Tuned_{CG}$ configurations of SCIP

Numerical results. We now report the numerical results when solving the integer problem. We compare using the (HPL) formulation and the (HPDW) formulation using the best configurations of the Branch-and-Price algorithm.

In Figures 5 and 6 we report the performance profile graphs for the two formulations where the y-axis represents the number of instances solved within the corresponding time (in seconds in logarithmic scale) in the x-axis, such that the higher the curve the better the formulation is performing.

The output data such as the CPU times (Times), the number of nodes and constraints (resp. Nodes and Cols), and the number of pricer iterations (Iters) are given of Tables 3 and 4 for respectively the resolution of the A1 instances and the complete graph instances. For each set of 10 instances not entirely solved, the number i of instances effectively solved is noted ^{*i} right next to the related CPU time. The two tables report the result of the instances which have been solved by both (HPL) and (HPDW).

From the graphs of Figures 5 and 6, we notice that the implementation of (HPL) solves more instances. However, for the A1 instances that we solved with both formulations, (HPDW) with the parameters tuned by `irace` is more efficient in terms of CPU time with the A1 instances than the (HPL). This isn't the case for one instance of the set ($a = 21$, $k = 91$), where (HPDW) have difficulties to find good variables to branch on. For this type

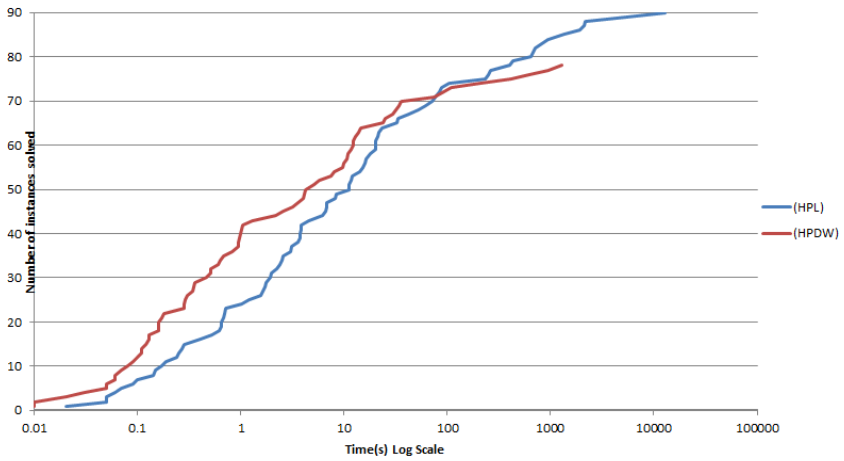

Figure 5: Performance Profile of (HPL) and (HPDW) on the A1 instances

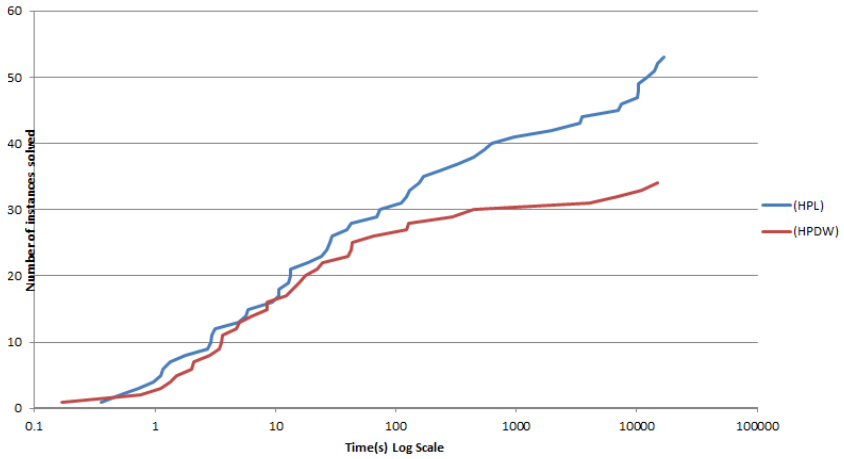

Figure 6: Performance Profile of (HPL) and (HPDW) on the complete graph instances

of real case instances, (HPDW) seems to be a good solution but this is not the case for pseudo-random generated data like the complete graph instances.

A1		21 commodities k			55 commodities k			91 commodities k		
(sets of 10 inst. for each ($\|\mathcal{K}\|$,$\|\mathcal{A}\|$))		20 a	56 a	90 a	20 a	56 a	90 a	20 a	56 a	90 a
(HPL)	Time(s)	0.12	1.16	1.88	2.32	13.16	28.33	13.05	276.26	672.44
	Nodes	16	86.8	32.1	141.10	489.30	485.10	1,327.50	9,199.00	9,740.00
(HPDW)	Time(s)	0.09	0.59	0.45	2.01	7.20	10.74	196.97^{*2}	141.02^{*6}	295.43^{*4}
"Tuned"	Nodes	54.4	152.3	56.3	232.30	831.40	649.40	3,151.75	4,502.00	10,103.17
	Cols	202.7	396.5	496.3	699.20	809.80	975.90	2,897.13	2,032.25	2,061.33
	Iters	71	145.3	58.6	388.50	856.80	664.90	3,550.75	4,255.00	9,156.33

Table 3: (HPL) Vs (HPDW) on the A1 instances

Complete Graph		20 commodities k			56 commodities k		
(sets of 10 inst. for each ($\|\mathcal{K}\|$,$\|\mathcal{A}\|$))		20 a	56 a	90 a	20 a	56 a	90 a
(HPL)	Time(s)	1.39	12.22	48.68	123.90^{*6}	–	–
	Nodes	198.50	465.50	711.10	16,830.50	–	–
(HPDW)	Time(s)	3.90	26.87	102.99	$9,217.16^{*6}$	–	–
"Tuned"	Nodes	1,149.90	3,889.30	9,096.00	20,294.25	–	–
	Cols	931.60	1,303.10	1,755.30	39,422.50	–	–
	Iters	1,192.00	3,764.60	8,616.40	28,891.25	–	–

Table 4: (HPL) Vs (HPDW) on the complete graph instances

We saw earlier that the efficiency of the (HPDW) on the complete graph instances are very sensitive to the different configurations. This formulation does not seem to be the right choice for solving this type of instance or, maybe, a more important work about the configuration must be done. With 20 commodities, (HPL) needs less than the half of the CPU time of (HPDW). The two formulations solve the same instances with 56 commodities and 20 tall paths. The half of them gives strong difficulties to (HPDW).

However, on A1 instances, which are more realistic for the NPP as they have an underlying network, there are some positive results, in particular on the 3 sets of instances with 20 and 56 commodities.

6. Conclusion

In this paper, we propose a new formulation for the Network Pricing Problem with Connected Toll Arcs. In previous work, this problem has been

modeled as a bi-level programming problem which has then been reformulated as a single-level linear programming problem denoted (HPL). This work presents a Dantzig-Wolfe reformulation (HPDW) of this problem. We prove that its linear relaxation, (HPDW-LR), is tighter than that of (HPL), denoted (HPL-LR), which means that F(HPDW-LR) $\subseteq F$(HPL-LR) where F is the set of feasible solutions of the corresponding formulation. Our computational experiments comply with our theoretical results.

In addition, we compare the resolution of (HPL) and (HPDW) experimentally through a Branch-and-Bound and Branch-and-Price algorithms, respectively. For some instances, it appears that (HPL) is more efficient (in terms of CPU time). Further, when comparing the number of nodes we obtain in both series of experiments, one explores more nodes solving the (HPDW) than (HPL), possibly due to the different branching strategies used in each approach. For (HPL), the framework *SCIP* automatically chooses the branching strategy. Recent updates to the framework indicate its improved efficiency over other frameworks. The branching strategy applied to (HPDW), on the other hand, has been developed manually. Improving upon this branching strategy will be tackled in future work.

Acknowledgements

This work is supported by the Interuniversity Attraction Poles Programme P7/36 "COMEX" initiated by the Belgian Science Policy Office and the BEWARE FELLOWSHIPS programs co-financed by the COFUND program of the European Union (FP7 - Marie Curie Actions).

Bibliography

A1.Instances, 2014 (Accessed). Profile of a1 highway milano - napoli. `http://www.automap.it/autostrade/mappa.asp?tratta=A1`.

Benichou, M., Gauthier, J., Girodet, P., Hentges, G., Ribiere, G., Vincent, O., 1971. Experiments in mixed-integer linear programming. Mathematical Programming 1 (1), 76–94.

Bracken, J., McGill, J., 1973. Mathematical programs with optimization problems in the constraints. Operations Research 21 (1), 37–44.

Candler, W., Norton, R., 1977. Multilevel programming. Tech. Rep. 20, World Bank Development Research Center, Washington DC, USA.

Colson, B., Marcotte, P., Savard, G., 2005. Bilevel programming: A survey. 4OR: A Quarterly Journal of Operations Research 3, 87–105.

Colson, B., Marcotte, P., Savard, G., 2007. An overview of bilevel optimization. Annals of Operations Research 153 (1), 235–256.

Dempe, S., 2002. Foundations of bilevel programming. Vol. 61 of Nonconvex optimization and its applications. Kluwer Academic Publishers.

Fortz, B., Labbé, M., Violin, A., 2013. Dantzig-wolfe reformulation for the network pricing problem with connected toll arcs. Electronic Notes in Discrete Mathematics 41 (0), 117 – 124.

Geoffrion, A., 1974. Lagrangean relaxation for integer programming. Approaches to Integer Programming, 82–114.

Hansen, P., Jaumard, B., Savard, G., 1992. A new branch-and-bound rules for linear bilevel programming. SIAM Journal on Scientific and Statistical Computing 5 (13), 1194–1217.

Heilporn, G., Labbé, M., Marcotte, P., Savard, G., 2010. A polyhedral study of the network pricing problem with connected toll arcs. Networks 3 (55), 234–246.

Heilporn, G., Labbé, M., Marcotte, P., Savard, G., 2011. Valid inequalities and branch-and-cut for the clique pricing problem. Discrete Optimization 8 (3), 393–410.

Jeroslow, R., 1985. The polynomial hierarchy and a simple model for competitive analysis. Mathematical Programming 32, 146–164.

Labbé, M., Marcotte, P., Savard, G., 1998. A bilevel model of taxation and its application to optimal highway pricing. Management Science 44 (12), 1608–1622.

Labbé, M., Violin, A., 2013. Bilevel programming and price setting problems. 4OR 11 (1), 1–30.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M., 2011. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium.

Lübbecke, M. E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53 (6), 1007–1023.

Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P., 1999. Stabilized column generation. Discrete Mathematics 194 (1-3), 229–237.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2013. In-out separation and column generation stabilization by dual price smoothing. In: Experimental Algorithms. Springer, pp. 354–365.

Pessoa, A., Uchoa, E., Aragão, M. P., Rodrigues, R., 2010. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation 2, 259–290.

Rosenberg, M. T., 2016.
URL `http://geography.about.com/library/weekly/aa031601a.htm`

Vicente, L., Calamai, P., 1994. Bilevel and multilevel programming: A bibliography review. Journal of Global Optimization 5, 291–306.

Vicente, L., Savard, G., Júdice, J., 1994. Descent approaches for quadratic bilevel programming. Journal of Optimization Theory and Applications 81 (2), 379–399.

Wentges, P., 1997. Weighted dantzig-wolfe decomposition for linear mixed-integer programming. International Transactions in Operational Research 4 (2), 151–162.