Università di Pisa

Dipartimento di Informatica

Technical Report: TR-07-08

Color-coding algorithms to the balanced path problem: computational issues

Paola Cappanera Maria Grazia Scutellà

March 13, 2007

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Color-coding algorithms to the balanced path problem: computational issues

Paola Cappanera   Maria Grazia Scutellà

March 13, 2007

Abstract

Given a weighted directed network G, we consider the problem of computing k *balanced* paths from given source nodes to given destination nodes of G, i.e., k paths such that the difference in cost between the longest and the shortest path is minimized.

Although not yet investigated by the OR scientific community, except for some preliminary theoretical results concerning the special case of acyclic networks, balanced path problems arise in several interesting applications, both in transportation and in telecommunication settings. For example, in load balancing of multiple Quality of Service (QoS) paths in the IP Telephony service, the problem of minimizing the end-to-end delay difference between the longest and the shortest path in multiple path management schemes has been suggested in the literature as a promising line of research.

In this work the focus is on the computation of node-disjoint balanced paths in the general case, where the input graph G may have any structure. Starting from some algorithmic ideas proposed for acyclic networks, a general framework is firstly described, which is based on the color-coding method for computing simple paths. Then the general framework is specialized and a pool of algorithms is designed which includes both an exact approach as well as alternative heuristics. The algorithms have been tested on a large suite of instances generated from some benchmark telecommunication instances.

The obtained computational results are very interesting: in some cases, the exact color-coding algorithm produces the optimal solution very rapidly; in the remaining cases, some of the proposed heuristics are able to generate high quality solutions in a very quick time.

1. Introduction

Let $G = (V, E)$ be a directed network, with $|V| = n$ and $|E| = m$, and c_{ij} be the nonnegative integral cost associated with (i, j), for each $(i, j) \in E$. Given k source nodes $\{s_1, s_2, \dots, s_k\}$ and k destination nodes $\{t_1, t_2, \dots, t_k\}$, $k \geq 2$, the *balanced path problem* on G is the problem of computing a set of k paths linking the k source nodes to the k destination nodes (there can be a path from any source to any destination, and all sources and destinations must be involved), in such a way that the difference in cost between the longest and the shortest path in the set is minimized. As usual, the cost of a path P is defined as the sum of the costs of the arcs in P. $\{s_1, s_2, \ldots, s_k\}$ are so ingress nodes of the network, which must be linked to the egress nodes $\{t_1, t_2, \ldots, t_k\}$ in a balanced way.

If we introduce two additional nodes s and t, such that s is connected to each source node s_i through a zero cost arc of type (s, s_i), and t is connected to each destination node t_i through a zero cost arc of type (t_i, t), then the balanced path problem can be equivalently formulated as the problem of computing k paths from s to t, involving the k arcs of type (s, s_i) and the k arcs of type (t_i, t), which are balanced according to the definition stated above. Throughout this paper we will impose that the computed balanced paths are node-disjoint. Similar algorithmic results can be obtained if arc-disjoint balanced paths have to be determined.

In addition to the above stated *single commodity version* of the problem, also a *multicommodity version* will be investigated. In such a case, given k origin-destination pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, $k \geq 2$, the problem consists of computing a set of k node-disjoint paths, linking the k origin-destination pairs, in such a way that the difference in cost between the longest and the shortest path in the set is minimized.

Balanced path problems arise in several interesting applications, both in transportation and in telecommunication frameworks. In the transportation area, the use of paths to represent weekly or monthly work assignments has been widely spread in crew scheduling problems; see for instance (Carraresi and Gallo 1984, Gamache et al. 1999, Cappanera and Gallo 2004). There, balancing aspects refer to an even distribution of the workload among the crew members. Other interesting applications can be found in the nurse rostering area (Burke et al. 2004). Furthermore, the balanced path computation can be a critical factor in addressing disruption problems in supply chain management. For example, some types of humanitarian goods, in emergency situations, need to be delivered along alternative paths, in such a way to minimize the difference in time between the late and the early arrival.

In the telecommunication area, disjoint paths are usually sought for network survavibility purposes. Here, balancing aspects may arise in routing "noninterfering" messages through a communication network. For example, Li, McCormick and Simchi-Levi (1990) addressed a quite related problem, consisting of computing two disjoint paths, from a super origin s to a super destination t on a weighted directed network G, guided by an objective function of type "min-max". Precisely, two disjoint paths from s to t have to be computed in such a way to minimize the cost of the longest path. In the balanced path problem, instead, the difference in cost between the longest and the shortest path has to be minimized. To the best of our knowledge, this kind of "balanced-type" objective function has never been investigated in the context of (disjoint) path computations, except for the preliminary theoretical results in Cappanera and Scutellà (2005), although different kinds of balanced optimization problems on graphs have already been studied by Martello et al. (1984), Camerini et al. (1986), Ahuja (1997) and Scutellà (1998).

An interesting application of the balanced path problem in the context of telecommunication networks is the load balancing of multiple Quality of Service (QoS) paths in the IP Telephony service, as described by Lee and Gerla (2001). There, different kinds of path management schemes are investigated, involving either single or multiple (partially) disjoint paths, to guarantee a good robustness against node and/or arc failures of the network. A key observation of the authors is that, in the case of schemes which spread packets over multiple disjoint paths, an important factor, which should be maintained as low as possible, is the end-to-end delay difference between the longest and the shortest path in each determined path set. This factor, which is related to the so-called jitter factor, can be critical especially in contexts such as video/audio conference interconnections and conduction of time critical experiments in a virtual collaboration. Minimizing the jitter factor over a set of disjoint paths naturally leads to the balanced path problem formulation, as previously introduced. Observe that balanced (disjoint) paths can be looked for also for some equity reasons, in order to guarantee an even response time for all the users of the network, when modelled in terms of given origin-destination pairs.

When node-disjoint (arc-disjoint) paths are looked for, the balanced path problem is strongly NP-Hard. This is true both in the single and in the multicommodity case. The strongly NP-Hardness result holds true also for the special case of acyclic networks (Cappanera and Scutellà, 2005).

Starting from some algorithmic ideas proposed for acyclic networks, in this paper we describe an algorithmic approach to solve the balanced path problem in the more general setting, considering both the single and the multicommodity scenarios. Since the approach exploits the idea of suitably coloring the nodes of the network to guarantee the computation of simple paths (Alon et al. 1995), it has been named the *color-coding approach*. Several algorithms have been derived from the color-coding framework. Specifically, the pool of color-coding algorithms includes both an exact algorithm, whose worst case time complexity figure is exponential with respect to the number of the nodes of the network, as well as alternative color-coding heuristics. The heuristics have been designed so as to overcome the inherent theoretical inefficiency of the exact algorithm. The objective of this paper is to validate the efficiency of the proposed color-coding algorithms from an empirical point of view. To this purpose, since no benchamark instances are available in the literature, we generated a large suite of balanced path instances starting from two sets of benchmark telecommunication instances. In particular, we maintained all the relevant information of the benchmark telecommunication instances (topology of the networks, routing costs, origin-destination information), so as to generate realistic balanced path telecommunication instances. So doing, the practical efficiency of the proposed methods has been investigated focusing on a telecommunication application context.

The rest of the paper is organized as follows. In Section 2 we propose multicommodity flow based formulations for the balanced path problem. The formulations have been used to validate, whenever possible, the quality of the solutions obtained by the proposed color-coding heuristics through direct comparison with the optimal solutions found by the commercial mixed-integer optimizer CPLEX (version 9.1). Specifically, the direct comparison was performed for those instances for which the exact color-coding algorithm failed in determining the optimum balanced path set. Section 3 describes the color-coding approach and the main algorithms which have been derived from it. Section 4 addresses the main implementation issues, while Section 5 reports the empirical results for the generated suite of balanced path instances. Finally, Section 6 presents conclusions and suggestions for future research.

2. Multicommodity flow based models

Here we introduce two ILP formulations for the balanced path problem (one formulation for the multicommodity scenario, and one formulation for the single commodity scenario). Before introducing the models we repute important to outline that alternative models might be proposed for the balanced path problem. Furthermore, probably the proposed formulations might be strengthened adding suitably defined valid inequalities. However, the line of research devoted to the study of alternative models to the problem, and to the proposal of efficient cutting plane algorithms based on these models, is out of the purpose of this paper which, as remarked in the Introduction, is to assess the computational efficiency of the proposed color-coding algorithms (exact and heuristic variants).

Let us first consider the version of the problem in which k origin-destination pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ are provided. In the case of acyclic networks, the balanced path problem can be formulated as the following nonlinear Multicommodity Flow Problem with node-disjoint constraints ($NDMCF$):

$$\min\Big(\max_h\{\sum_{(i,j)\in E} c_{ij}x_{ij}^h\} - \min_h\{\sum_{(i,j)\in E} c_{ij}x_{ij}^h\}\Big) \tag{1}$$

$$\text{s.t.} \sum_{(s_h,i)\in E} x_{s_h i}^h = 1 \qquad h = 1, .., k \tag{2}$$

$$\sum_{(i,t_h)\in E} x_{it_h}^h = 1 \qquad h = 1, .., k \tag{3}$$

$$\sum_{(j,i)\in E} x_{ji}^h - \sum_{(i,j)\in E} x_{ij}^h = 0 \qquad h = 1, .., k, \quad \forall i \in V \setminus \{s_h, t_h\} \tag{4}$$

$$\sum_{(i,j)\in E} \sum_h x_{ij}^h \leq 1 \qquad \forall i \in V \tag{5}$$

$$x_{ij}^h \in \{0, 1\} \qquad \forall (i,j) \in E, \quad h = 1, .., k \tag{6}$$

In the formulation, commodity h is used to describe the path linking the origin-destination pair (s_h, t_h), $h = 1, \ldots k$, and x_{ij}^h thus represents the flow along the arc (i, j) relative to commodity h. Constraints (2),(3),(4) are the classical flow conservation constraints which assure that, for each commodity h, a unitary flow is pushed from the origin s_h to the destination t_h. Constraints (5) are the node-disjoint constraints, which guarantee that each node is considered at most once.

The objective function (1), which is expressed in a nonlinear form, provides the difference between the maximum and the minimum commodity cost. Such a function can be linearized in a standard way by introducing two variables $z_{\max}$ and $z_{\min}$ which bound, respectively from above and from below, the maximum and the minimum cost of the k paths:

$$
\begin{array}{lll}
\min & z_{\max} - z_{\min} & \\
\text{s.t.} & x^h \text{ satisfies } (2)-(6) & h = 1,..,k \\
& \sum_{(i,j)\in E} c_{ij} x_{ij}^h \leq z_{\max} & h = 1,..,k \\
& \sum_{(i,j)\in E} c_{ij} x_{ij}^h \geq z_{\min} & h = 1,..,k
\end{array}
$$

Indeed, since general networks are considered, constraints (2),(3),(4) and (5), together with the integrality of the flow variables, are not sufficient to guarantee that the solution of the model is a set of k paths. In fact, in order to reduce the objective function value, some commodities, say h, could send a unitary flow along one or more directed cycles of the network, in addition to the unitary flow sent along a directed path from the source s_h to the destination t_h. So, the above stated formulation must be enhanced in order to prevent this kind of infeasible solutions.

To this end, let us consider a set of additional constraints which have the same structure of the subtour elimination constraints used for formulating the classical traveling salesman problem (*TSP*). Several families of subtour elimination constraints have been proposed in the literature whose cardinality, unfortunately, can be exponential with respect to the problem size; these constraints are in fact related to all possible subsets of nodes in the network. However, alternative subtour elimination constraints have been suggested, having a polynomial cardinality (Miller et al. 1960). In the context of *TSP*, such constraints are usually formulated as follows:

$$u_i - u_j + n x_{ij} \leq n - 1 \quad \forall (i,j) \in E,\ i \neq 1\ j \neq 1 \tag{7}$$

In (7), n denotes the cardinality of the node set, x_{ij} is the binary variable indicating whether arc (i,j) belongs to the tour, while node 1 is assumed to be the depot node the tour must begin from and end to. Variables $u \in R^{n-1}$ are associated with the nodes and they can be interpreted as the position of a node in the tour.

Constraints (7) can be specialized to the balanced path problem. In this case, the node-disjointness is explicitly guaranteed by constraints (5). However, as observed before, it may happen that the solution relative to a commodity h contains one or more directed cycles in addition to a directed path. Such cycles surely do not involve the origin s_h and the destination

t_h of such a commodity. We can thus exclude these cycles by adding the following constraints to the formulation previously introduced:

$$u_i - u_j + n x_{ij}^h \leq n - 1 \quad \forall (i,j) \in E,\ h = 1,..,k,\ i,j \notin \{s_h, t_h\} \tag{8}$$

Such constraints cut off all subtours not including nodes $s_1, \ldots, s_k, t_1, \ldots, t_k$. Consider now any feasible set of node-disjoint paths linking the origin-destination pairs $\{(s_1,\ t_1), \ldots, (s_k, t_k)\}$. As for (*TSP*), assign to variable u_i the position of node i in a directed path of the considered solution, if i belongs to a path composing such a solution; otherwise, set u_i to the maximum value assigned to the u variables associated with nodes composing the feasible set of paths. It is easy to see that (8) are satisfied both by each arc (i,j) composing the given feasible set of paths, and also by each unused arc (not involving $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$). In the last case, (8) are redundant since the involved variables u_i and u_j assume a value in the range between 2 and $n - 2k + 1$. In fact, from the node-disjointness of the paths, the cardinality of each feasible path is bounded from above by $n - 2k + 1$. The formulation to the multicommodity version of the balanced path problem, (mBP), is therefore:

$$\begin{array}{lll}
\min & z_{\max} - z_{\min} & \\
\text{s.t.} & \displaystyle\sum_{(s_h,i)\in E} x_{s_h i}^h = 1 & h = 1,..,k \\
& \displaystyle\sum_{(i,t_h)\in E} x_{i t_h}^h = 1 & h = 1,..,k \\
& \displaystyle\sum_{(j,i)\in E} x_{ji}^h - \sum_{(i,j)\in E} x_{ij}^h = 0 & h = 1,..,k, \qquad \forall i \in V \setminus \{s_h, t_h\} \\
& \displaystyle\sum_{(i,j)\in E} \sum_{h} x_{ij}^h \leq 1 & \forall i \in V \\
& u_i - u_j + n x_{ij}^h \leq n - 1 & \forall (i,j) \in E, \qquad h = 1,..,k, i, j \notin \{s_h, t_h\} \\
& \displaystyle\sum_{(i,j)\in E} c_{ij} x_{ij}^h \leq z_{\max} & h = 1,..,k \\
& \displaystyle\sum_{(i,j)\in E} c_{ij} x_{ij}^h \geq z_{\min} & h = 1,..,k \\
& x_{ij}^h \in \{0,1\} & \forall (i,j) \in E, \qquad h = 1,..,k
\end{array}$$

When the single commodity version of the problem is addressed, i.e., each source can be linked to any destination, we can modify the flow conservation constraints (2),(3),(4) as follows. Consider the equivalent formulation in which an additional source s and an additional destination t are introduced, such that s is connected to each s_i through a zero cost arc of type (s, s_i), and t is connected to each t_i through a zero cost arc of type (t_i, t). Then, it is sufficient to impose that s and t are, respectively, the source and the destination of each commodity, requiring a unitary flow. Furthermore, the node-disjointness is imposed for each node other than s and t. Concerning constraints (8), they can be modified in a straightforward way as follows:

$$u_i - u_j + nx_{ij}^h \leq n - 1 \quad \forall (i,j) \in E,\ h = 1,..,k,\ i,j \notin \{s, t, s_1, ., s_k, t_1, ., t_k\} \tag{9}$$

The obtained formulation, (sBP), is therefore:

$$\begin{array}{lll}
\min & z_{\max} - z_{\min} & \\
\text{s.t.} & \sum_{(s,i)\in E} x_{si}^h = 1 & h = 1,..,k \\
& \sum_{(i,t)\in E} x_{it}^h = 1 & h = 1,..,k \\
& \sum_{(j,i)\in E} x_{ji}^h - \sum_{(i,j)\in E} x_{ij}^h = 0, & h = 1,..,k, \quad \forall i \in V \setminus \{s,t\} \\
& \sum_{(i,j)\in E} \sum_{h} x_{ij}^h \leq 1, & \forall i \in V \setminus \{s,t\} \\
& u_i - u_j + nx_{ij}^h \leq n - 1 & \forall (i,j), \quad h, \quad i,j \notin \{s, t, s_1, ., s_k, t_1, ., t_k\} \\
& \sum_{(i,j)\in E} c_{ij} x_{ij}^h \leq z_{\max} & h = 1,..,k \\
& \sum_{(i,j)\in E} c_{ij} x_{ij}^h \geq z_{\min} & h = 1,..,k \\
& x_{ij}^h \in \{0,1\} & \forall (i,j) \in E, \quad h = 1,..,k
\end{array}$$

Indeed, since paths are node-disjoint, then constraints (8) and (9) can be aggregated by commodity and tightened as follows (in (10), s and t appear only in the single commodity formulation):

$$u_i - u_j + n\sum_{h=1}^{k} x_{ij}^h \leq n - 1 \quad \forall (i,j) \in E,\ i,j \notin \{s, t, s_1, \ldots, s_k, t_1, \ldots, t_k\} \tag{10}$$

Furthermore, from the above observations concerning the values assumable by variables u_i in any feasible solution, then the coefficient of $\sum_{h=1}^{k} x_{ij}^h$ and the right hand side of constraints (10) can be tightned to $n - 2k$ and to $n - 2k - 1$, respectively.

3. The color-coding algorithms

Here we describe the general color-coding framework and the main algorithms which have been designed starting from this framework.

3.1. The color-coding algorithmic framework

Let us consider the single commodity version of the balanced path problem, making use of a super source s, linked to the k original source nodes, and of a super destination t, linked to the original destination nodes. Firstly assume that no disjointness constraint is imposed on the paths to be balanced, and U denote an upper bound to the cost of the longest path in G.

Given U, let us introduce the *cost-expanded network* $G_U = (V_U, E_U)$, that is the unweighted layered network obtained from G by inserting s as well as $s_1 \dots , s_k$ in V_U, and replacing each other node i of G by $U + 1$ copies i^p, $p = 0, 1, 2, \dots , U$, where i^p represents node i when reached from the source s in G through paths of cost exactly p, $p = 0, 1, 2, \dots , U$. Each arc (i, j) of G may therefore generate several arc copies in G_U, that is copies of type $(i^p, j^{p+c_{ij}})$, for each feasible index p. By construction, each path from s to t in G of cost p is mapped onto a directed path of G_U from node s to node t^p (i.e., to the copy t^p of the destination node t). Observe that G_U is a layered network having $O(nU)$ nodes and $O(mU)$ arcs: its size is therefore pseudo-polynomial with respect to the size of the original network G.

In order to solve the balanced path problem on G, let us perform a binary search in the interval $[0, U]$. For each considered integer value δ in $[0, U]$, the approach constructs the family of subgraphs of G_U composed by the paths of G_U going from node s to the nodes in $\{t^p, \dots , t^{p+\delta}\}$, for each suitable p, and verifies whether at least one of these subgraphs contains k distinct paths. At the termination, the last returned set of paths constitute an optimum (balanced) set of paths, and the corresponding δ is the optimum objective function value (i.e., the minimum cost difference among the paths).

As shown in (Cappanera and Scutellà 2005) for the case of acyclic networks, when node-disjointness is relaxed, then the check in each subgraph of G_U can be performed via a suitable maximum flow computation. The approach thus checks $O(\log U)$ values in $[0, U]$ and, for each considered value δ, it computes a maximum flow in $O(U)$ networks having $O(mU)$ arcs. Since the maximum flow value is at most k, then each maximum flow can be computed in $O(kmU)$ time. The proposed approach therefore determines an optimum solution in $O(kmU^2 \log U)$ time.

It is known (Cappanera and Scutellà 2005) that when the node-disjointness of the k paths to be balanced is required, then the problem is strongly NP-Hard also for acyclic networks (same considerations hold for arc-disjoint paths). In such a case, during the binary search we impose the node-disjointness of the computed paths by suitably adapting the color-coding method for computing simple paths (Alon et al. 1995).

Let Col denote a set of ν colors (ν is a parameter of the approach), and R indicate the set of random colorings of the nodes of G with exactly ν colors. Let r be a random coloring in R. Define a set of k paths from s to t in G *colorful* if the nodes composing these paths are colored by distinct colors under r.

Given r, the idea is to apply the pseudo-polynomial approach previously described (proposed for the case where no disjointness constraint is imposed) by using the coloring of the nodes as a means to guarantee, in each considered subgraph of the cost-expanded network G_U, the computation of node-disjoint paths (remember that several copies of any node i can be present in G_U). More precisely, extend the coloring r to the nodes of G_U in such a way that copies of a same node i in G_U are colored in the same way, that is by the same color of i in G under the coloring r. Then, apply a variant of the pseudo-polynomial approach where, for each considered subgraph of G_U generated during the binary search in the interval $[0, U]$, k node-disjoint paths from s to t are looked for. To this end, we look for a colorful set of k paths in the considered subgraph; that is, we use the coloring r to guarantee the computation of k paths which are node-disjoint (with respect to r).

It is straightforward to show that a colorful set of k paths can be modelled as a unique, colorful path from s to t of cardinality at most ν. To this end, it is sufficient to add arcs linking each (true) destination node to each (true) source node, and to look for a colorful path from s to t containing $k-1$ of these additional arcs. This can be performed by assigning cost 1 to each additional arc, cost 0 to each original arc, and looking for a colorful path from s to t of maximum cost: a colorful set of k paths exists if and only if the returned maximum cost is equal to $(k-1)$.

We can look for a path of this kind by the method proposed by Alon et al. (1995) for computing simple maximum cost paths of fixed cardinality in a directed graph. The computation is performed in $O(2^h mU)$ time, where h is $O(\nu)$ (recall that mU bounds from above the cardinality of the arc set of G_U). Given a random coloring $r \in R$, the approach is therefore able to determine a balanced colorful set of paths, with respect to r, in $O(2^h m U^2 \log U)$ time, where h is $O(\nu)$.

Given a subset of R, the *color-coding approach* computes a balanced colorful set of paths with respect to each coloring in the chosen subset, and returns the best computed solution.

The color-coding approach can be easily modified in order to solve, with the same time complexity, the case of given origin-destination pairs. In fact, in such a multicommodity flow scenario, it is sufficient to look for colorful paths from a fictitious source s to a fictitious destination t in a transformed graph where s is linked only to s_1, t_1 is linked to s_2, t_2 is linked to s_3 and so on, and where only node t_k is linked to the (fictitious) destination t. In this way, at each iteration the approach looks for a colorful path from s to t which includes, in this order, the arcs (t_1, s_2), ... , (t_{k-1}, s_k) (the other colorful paths, considered infeasible, are disregarded during the visit of the graph), so generating sets of paths linking the given origin-destination pairs.

3.2. Exact and heuristic color-coding algorithms

Starting from the color-coding framework, several algorithms have been proposed to solve the balanced path problem. We report here about the ones that produced the best computational results, together with some related properties. This pool of algorithms consists of an exact algorithm, which has an exponential worst case time complexity, and three heuristics designed in an attempt to reduce the computational burden of the exact approach. Let us describe the algorithms in more details.

The exact color-coding algorithm

The algorithm is based on the following observation. Since the cardinality of the balanced path solution is bounded from above by n, then the setting $\nu = n$ in the color-coding framework allows one to derive an *exact color-coding algorithm* for the balanced path computation. By this setting, a single random coloring of the nodes of G is clearly sufficient to discover the optimum solution.

As follows from t1he time complexity analysis of the color-coding framework, the exact algorithm has a worst case time complexity which is exponential with respect to the number n of nodes of G (remember that the node-disjoint balanced path problem is strongly NP-Hard). Furthermore, such a time complexity quadratically increases with U, i.e., the estimate of the longest path cost of G.

In an attempt to reduce the computational burden of the exact color-coding algorithm, especially when n and U increase, three heuristics have been investigated which are based on the following tools: (i) the use of a number of colors $\nu < n$; (ii) a time limit in the binary search performed by the color-coding framework; and (iii) the use of a suitable cost scaling device, in order to reduce the size of the cost-expanded graph.

The ν-heuristic

By setting a number of colors $\nu < n$ in the color-coding framework, the approach turns out into a heuristic, namely the *ν-heuristic*. Specifically, given $\nu < n$, the ν-heuristic generates a list of random colorings with exactly ν colors, iterates the color-coding framework for each coloring and and returns the best solution among the ones found for each coloring in the generated list.

The behavior of the ν-heuristic can be estimated from a probabilistic point of view. To this end, let introduce the following definition:

Definition 3.1 *Let r_1 and r_2 be random colorings in R with exactly ν colors. r_1 and r_2 are said to be* equivalent *if there exists a bijection $\lambda : Col \to Col$ such that $r_2 = \lambda \bullet r_1$.*

In other words, r_1 and r_2 are equivalent colorings when they induce the same partition of the nodes of G, and so any subgraph is colorful with respect to r_1 if and only if it is colorful with respect to r_2.

Now, let us denote by Π_ν^n the number of the partitions of a set of n elements into ν (not empty) subsets. From the above considerations, this provides the total number of non equivalent random colorings of the nodes of G with exactly ν colors. As a consequence, if there exists a balanced path solution which is formed by $O(\nu)$ nodes, and if only non equivalent random colorings are generated, then the ν-heuristic is able to determine an optimum solution with probability $\frac{1}{\Pi_\nu^n}$. In order to explicitate Π_ν^n, observe that, since the total number of random colorings using at most ν colors is ν^n, then the total number of random colorings using exactly ν colors is $\nu^n - (\nu-1)^n$. Therefore, since the number of random colorings which are equivalent to a given random coloring r is $\nu!$, then $\frac{\nu^n-(\nu-1)^n}{\nu!}$ gives the total number of non equivalent random colorings with exactly ν colors, that is Π_ν^n. Expression $\frac{\nu!}{\nu^n-(\nu-1)^n}$ thus provides the above stated probability. If α independent random colorings are generated, then the probability becomes $\frac{\alpha\nu!}{\nu^n-(\nu-1)^n}$. Then:

Property 3.1 *If there exists a balanced path solution which is formed by $O(\nu)$ nodes, and if α non equivalent random colorings of ν colors are generated, then the ν-heuristic is able to determine an optimum solution with probability $\frac{\alpha\nu!}{\nu^n-(\nu-1)^n}$.*

As discussed in Section 4, all the implemented color-coding heuristics are equipped with sophisticated devices aimed at generating non equivalent random colorings. The above stated property thus holds true, providing the probabilistic behavior of the ν-heuristic.

The 5+5 heuristic

The *5+5* heuristic is based on the following observation. In performing the binary search, it may happen that the color-coding approach get trapped into an semi-interval of the binary search. Since a time limit of one hour was imposed for each algorithm execution during the computational experimentation, that would be critical, since a (possibly large) portion of the space of the feasible solutions could be excluded from the search. To avoid that, a time limit of 5 minutes is imposed on the time elapsed from an improvement to the next one. When this time limit is reached the search is stopped, and the heuristic jumps to the next range of the binary search interval. After two trials (*5+5*) the *5+5* heuristic stops, returning the best computed solution.

The cost scaling heuristic

In order to reduce the size of the cost-expanded graph, and consequently produce better performances, the following cost transformation is performed before the execution of the binary search. Given a positive constant ϵ, scale the original arc costs c_{ij} so they live in the range $0 \dots n/\epsilon$, and then truncate them to the nearest integer. That is

$$c'_{ij} = \left\lceil \frac{c_{ij}}{c_{max}} \frac{n}{\epsilon} \right\rceil,$$

where c_{max} denotes the maximum arc cost of network G. The integer truncation causes each of the rescaled arc costs to change in value by at most 1, so each path cost changes in value by at most n.

The *cost scaling* heuristic applies the color-coding approach using the transformed costs c'_{ij} instead of the original arc costs c_{ij}, $\forall (i,j) \in E$. As an effect of the cost scaling, the size of the cost-expanded graph is now a non-increasing function of parameter ϵ; in fact, the cost-expanded graph is now composed of $O(\frac{n^2}{\epsilon})$ nodes and of $O(\frac{mn}{\epsilon})$ arcs. Setting the number of colors $\nu = n$, the *cost scaling* heuristic therefore looks for a balanced colorful set of paths in $O(2^n m(\frac{n}{\epsilon})^2 \log(\frac{n}{\epsilon}))$ time.

It is easy to prove that the absolute error committed by the cost scaling heuristic can be bounded from above by (ϵc_{max}). Therefore:

Property 3.2 *The cost scaling heuristic determines an approximate solution, having an absolute error of $O(\epsilon c_{max})$, in $O(2^n m(\frac{n}{\epsilon})^2 \log(\frac{n}{\epsilon}))$ time.*

As shown in Section 5, the cost scaling heuristic proved to be very effective in computing good quality solutions, which very often qualified as the optimum solutions.

4. Some implementation issues

Both the exact and the heuristic color-coding algorithms adopt the following described implementation issues. In this section s and t will be used to denote both the super origin and the super destination in the single commodity version of the problem, and also the fictitious source and the fictitious destination, as described in Section 3.1, in the multicommodity context.

4.1. The upper bound U

Value U, which bounds from above the cost of the longest path in G, and which determines the size of the cost-expanded graph G_U, is set by all algorithms to the value $(n-2k+1)c_{max}$. In fact, since node-disjoint paths involving $\{s_1, s_2, \ldots, s_k\}$ and $\{t_1, t_2, \ldots, t_k\}$ are formed by at most $(n-2k+1)$ arcs, such a value estimates from above the cost of the longest path in G. Observe that, in the cost scaling heuristic, c_{max} refers to the maximum arc cost after cost scaling.

4.2. The binary search interval

Once generated G_U, the width of the binary search interval is estimated according to the following observations. Let $\{t^{\alpha_{min}}, \ldots, t^{\alpha_{max}}\}$ denote the copies of the destination t in G_U. Then, it is easy to observe that the difference in cost between the longest and the shortest paths in G is bounded from above by $\Delta = (\alpha_{max} - \alpha_{min})$. The binary search can therefore be performed in the interval $[0, \Delta]$. Clearly Δ is less than or equal to the rough estimate

U. Indeed, for several balanced path instances the reduction of the binary search interval so obtained was substantial, as reported in Section 5.

A further reduction of the binary search interval may be obtained by looking for an *initial feasible solution*. In the single commmodity case, the algorithms solve a minimum cost flow problem in a transformed network G' having $\{s_1, s_2, \ldots, s_k\}$ as the source set, and $\{t_1, t_2, \ldots, t_k\}$ as the destination set. Each node s_i is a source of a unitary flow, and each node t_i is a destination of a unitary flow, $i = 1, \ldots, k$. G' is obtained from the original network G as follows. Duplicate each node i of G, and denote by i^1 and i^2 the obtained copies; direct each arc of G of type (j, i) into i^1, and direct each arc of G of type (i, j) out of i^2, by maintaining the original arc costs. Finally, insert a zero cost arc (i^1, i^2) between each pair of duplicated nodes. By construction, node-disjoint paths in G are in one-to-one correspondence with arc-disjoint paths in the transformed network G'. Then, in order to impose the arc-disjointness of the paths in G', assign a unit capacity to each arc of G' (indeed, we can limit this constraint only to the arcs of form $(i^1,\ i^2)$). Once computed a minimum cost flow in G', the corresponding set of arc-disjoint paths always provides a feasible solution to the problem. In fact, it represents a set of node-disjoint paths of G going from $\{s_1, s_2, \ldots, s_k\}$ to $\{t_1, t_2, \ldots, t_k\}$. If δ^* denotes the cost difference among these paths, and if $\delta^* - 1 < \Delta$, then the binary search can be performed in the reduced interval $[0, \delta^* - 1]$.

In the multicommodity case, once established an arbitrary ordering of the origin-destination pairs, an initial feasible solution is looked for by iteratively computing a shortest path for each commodity with respect to the chosen ordering, each time inhibiting the nodes of the computed shortest path from the next computations. If a set of k paths is found, and if $\delta^* - 1 < \Delta$ where, again, δ^* denotes the cost difference among the computed paths, then the binary search can be performed in the reduced interval $[0, \delta^* - 1]$. In our approach we generated different orderings of the origin-destination pairs, and returned the best solution found (if at least one fesible solution is determined).

In all the tested networks, and both in the single commodity and in the multicommodity versions, the procedures aimed at computing an initial feasible solution proved to be very fast and very effective. Specifically, they allowed to reduce the binary search interval $[0, \Delta]$ by the 84% on the average in the single commodity case, and by the 78,6% on the average in the multicommodity scenario.

4.3. Generation of non equivalent random colorings

The color-coding heuristics are equipped with suitable devices aimed at avoiding the generation of equivalent colorings of the nodes, according to Definition 3.1.

This is obtained by representing the partition of the nodes, induced by a random coloring, via techniques which are similar to techniques adopted in some search engines in order to classify documents and perform queries related to these documents. More in detail, the node partition induced by a random coloring is represented in such a way that each subset in the partition is implemented as an ordered set of nodes, where the first node has the role of a "pivot". When two colorings, say r_1 and r_2, are compared in order to verify their equivalence, the subsets of nodes induced by r_1 and r_2 are compared by performing a first comparison concerning the pivots, and then, if such a comparison succeeds, by suitably comparing the ordered subsets of the nodes.

4.4. The generated subgraphs and related properties

For each considered δ in the binary search interval all the color-coding algorithms construct the family of the subgraphs of G_U composed by the paths of G_U going from node s to nodes in $\{t^p, \ldots, t^{p+\delta}\}$, for each suitable p, and verify whether at least one of these subgraphs contains a colorful path from s to t. In our implementation, a careful generation of this family of subgraphs has been performed. The aim is to avoid the generation of subgraphs which surely do not contain a colorful path (for example, they do not include copies of a certain destination t_i). Moreover, also the generation of redundant subgraphs is pursued. For example, some sets $\{t^p, \ldots, t^{p+\delta}\}$ may appear in more subgraphs of G_U; in such a case, it is sufficient to generate only one of these subgraphs.

Furthermore, given δ, while looking for a set of node-disjoint paths with cost difference $\leq \delta$, the algorithms give priority to the sets composed of "shortest" paths. Such a tool is implemented by generating the subgraphs composed of paths from s to nodes in $\{t^p, \ldots, t^{p+\delta}\}$, for suitable values of p, in nondecreasing order of values of p. So doing, an interesting property of the exact color-coding algorithm is that it generates a set of balanced paths which minimizes the cost of the longest path (with respect to the set of the optimum, balanced, solutions). This is relevant in several application contexts, for example in telecommunication applications, where "short" paths are anyway preferred.

Property 4.1 *The exact color-coding algorithm generates a set of balanced paths which minimizes the cost of the longest path.*

This aspect will be better discussed in Section 5, devoted to the presentation of the computational investigation and to the analysis of the empirical results. In particular, it will be shown that the disjoint paths computed by the color-coding algorithms are quite "short" on the average, both considering the maximum hop cardinality and the longest path cost. So, the generated paths well satisfy these Quality of Service requirements. These requirements are particularly relevant in the context of telecommunications, on which we put the emphasis during the empirical investigation.

4.5. The colorful path computation

All the algorithms look for a colorful path from s to t by performing a suitable dynamic programming approach, along the lines of the classical dynamic programming approach for constrained shortest path computations (Desrochers and Soumis 1988). In the balanced path context, the peculiarity of the dynamic programming approach is that the consumption of the resource "color" has to be taken into account during the generation of the paths. In the proposed implementation, the labels associated with the nodes of the cost-expanded graph are bit arrays of size ν, which are used to take trace of the colors painting the nodes of all visited paths. Suitable logic operations are then performed to efficiently update these bit array labels during the dynamic programming computation.

5. The computational study

In this section we report the computational results obtained for a large suite of balanced path instances. The instances have been generated starting from two sets of benchmark telecommunication instances. Firstly the two data sets will be described. Then, the obtained empirical results will be presented, separately for the two families of balanced path instances. Finally, interesting QoS properties of the generated solutions will be discussed.

5.1. The data sets

Two data sets have been generated starting respectively from 29 and 52 telecommunication instances available at:

- the ZIB (`http://sndlib.zib.de`)
- the Politecnico of Milano

  (`ftp://ftp.elet.polimi.it/users/Pietro.Belotti/mcf/data`).

The first group of instances will be referred to as *Sndlib* instances, while the second one will be referred to as *Belotti* instances. For each original instance, two balanced path instances have been generated, related respectively to the single and to the multicommodity scenario. In order to distinguish the two cases, an F is appended to the instance name in the single commodity scenario, while, in the multicommodity case, a T is used instead. $k = \lceil \log n \rceil$ ingress and egress nodes and $k = \lceil \log n \rceil$ origin-destination pairs were generated for the instances of type F and of type T, respectively, starting from origin-destination information of the original telecommunication instances.

In more details, as far as the SndLib data set is concerned, the generated instances preserve the same structure of the original networks and exploit the routing costs, where present, as the network arc costs. Since some original instances are characterized by zero arc costs, in the instance generation phase an integer randomly chosen in the range [1,50] has been added to each original arc cost. Origin and destination nodes have been selected by considering the cardinality of their backward and forward star respectively: in particular, the origin nodes are the first k nodes selected in a nondecreasing order of the backward star cardinality; simmetrically, the destination nodes are chosen as the first k nodes in a nondecreasing order of the forward star cardinality.

On the other hand the Belotti instances, which are related to fiber-optic telecommunication network design problems and which exhibit a multicommodity flow network structure, maintain the same structure as the original instances and the original arc routing costs. Furthermore, k origin-destination pairs among the ones listed in the original files are selected to define the k ingress nodes and the k egress nodes (for the instances of type F) and the k origin-destination pairs (for the instances of type T). Since the original networks are undirected, we transformed them into directed networks by duplicating each original arc and assigning each copy the same cost as the original arc. It is remarkable to observe that the networks thus obtained are strongly cyclic.

The details of the instances are reported in Tables 1 and 2 in terms of number of nodes n, number of arcs m, number of paths k to be balanced, estimate of the longest path cost U, as

Table 1: SndLib data set

ProbName	n	m	k	U	Δ
atlanta	15	22	3	1150	413
dfn-bwin	10	45	3	250	181
dfn-gwin	11	47	3	300	226
di-yuan	11	42	3	300	292
france	25	46	4	900	884
italy	30	59	4	1150	479
janos-eu	37	114	5	1400	1349
janos-us	26	84	4	950	911
jano-us-ca	39	122	5	1500	1480
newyork	16	49	4	468	209
nobel-eu	28	41	4	1050	138
nobel-germany	17	26	4	480	72
nobel-us	14	21	3	432	79
norway	27	51	4	1000	261
nsfnet	14	21	3	432	72
otn-ibcn-ibbt	37	57	5	56448	2651
pdh1	11	34	3	300	190
pdh2	11	34	3	300	190
pioro40	40	89	5	1550	209
polska	12	18	3	336	57
rome13	14	21	3	432	82
sdh1	18	33	4	550	119
sun	27	102	4	1000	980
ta1	24	55	4	850	816
ta2	65	108	6	2700	246
zib16	16	51	4	450	177
zib17	17	31	4	500	122
zib28	28	41	4	1050	123
zib54	54	81	5	2250	106

defined in Section 4.1, and upper bound Δ of the binary search interval width, as described in Section 4.2.

5.2. The computational results

We begin this section by giving some details on the way the computational experience was performed and illustrating our choice for reporting the computational results. To begin with, for each instance in the two data sets first we ran the exact color-coding algorithm, i.e., the color-coding framework where: (i) the number of used colors ν is equal to n and, consequently, (ii) only one coloring is used. This algorithm will be referred to as *ExactCC*. A time limit (TL) of one hour of CPU time was imposed both for ExactCC as well as for each tested color-coding algorithm.

The purpose was to try to characterize the overall behavior of the exact color-coding algorithm, in an attempt to classify the instances into "easy" or "difficult" ones. In the following sections, for the two data sets separately, we report the number of times ExactCC rapidly determined the optimum solution, within 10 seconds. Since these aggregated results

Table 2: Belotti data set

ProbName	n	m	k	U	Δ
arpa2_99	21	52	4	266	14
arpanet_99	24	100	4	1530	1520
bhv1c	14	38	3	3951	3706
bhv2c	24	56	4	9044	8470
bhv3c	29	124	4	12342	12270
bhv4c	18	60	4	4829	4717
bhv5c	19	54	4	6732	6571
bhv6c	27	78	4	10460	10191
bhv7c	23	66	4	8384	8205
bvh8c	28	70	4	11151	10944
bvh9c	24	86	4	7616	7482
bhvac	19	46	4	5268	5085
bhvbc	14	36	3	3960	3856
bhvcc	27	78	4	12380	12150
bhvdc	29	72	4	10208	9743
bhvec	20	54	4	5889	5801
cost239	11	44	3	144	139
eon_5	19	74	4	420	415
eon_25	19	74	4	420	413
latadl_5	39	172	5	1950	1933
latadl_25	39	172	5	1950	1921
latadl_99	39	172	5	1950	1940
net_15_5	15	74	3	5500	5290
net_15_20	15	70	3	5560	5474
net_15_60	15	74	3	5020	4989
net_15_99	15	64	3	4680	441
net_20_5	20	96	4	6305	6270
net_20_20	20	100	4	7267	7160
net_20_60	20	92	4	6253	6119
net_20_99	20	98	4	6071	6003
net_25_5	25	118	4	7740	7610
net_25_20	25	116	4	8316	8226
net_25_60	25	124	4	9018	8958
net_25_99	25	124	4	10548	10431
net_30_5	30	142	4	11753	11482
net_30_20	30	142	4	13685	13500
net_30_60	30	146	4	12052	11935
net_30_99	30	146	4	12627	12605
njlata_5	11	46	3	570	523
njlata_25	11	46	3	552	517
njlata_99	11	46	3	570	540
nsf1a	14	42	3	1440	1400
nsf1b	14	42	3	1440	1390
nsf2	14	44	3	1440	1400
pacbell_25	15	42	3	300	289
pacbell_99	15	42	3	300	291
toronto_5	25	110	4	162	160
toronto_25	25	110	4	162	160
toronto_99	25	110	4	162	160
usld_5	28	90	4	1302	1270
usld_25	28	90	4	1302	1279
usld_99	28	90	4	1302	1270

seem to be encouraging and show that the exact color-coding algorithm alone is able to obtain quite good performances at least on a meaningful number of instances, we decided to focus on much clever implementations of the color-coding framework just to cope with the most critical instances. In an attempt to reduce the computational burden of ExactCC on such instances, the three color-coding heuristics described in Section 3.2 have been implemented. We recall that they are based on the following tools: (i) an additional time limit which results in the *5+5* heuristic; (ii) use of a number of colors $\nu < n$ which results in the ν-heuristic; and (iii) use of the cost scaling device (*cost scaling* heuristic). Observe that these three tools could be combined together. Indeed, in our computational investigation we experimented some combinations, observing that often the joint use of two or three of such tools may result in much better performances than the ones obtained when only one tool is considered. However, we preferred to report here the gain or loss in efficiency of the three tools separately, so that their effect and contribution may result more clear to the reader.

On the most critical instances generally ExactCC was not able to certify the optimality of the obtained solution within the stated time limit of one hour. We thus used the commercial mixed-integer optimizer CPLEX (version 9.1), on those critical cases, to validate the quality of the solutions computed by the color-coding heuristics, and to have an idea of their computational efficiency. CPLEX ran model (mBP) in the case of given origin-destination pairs (i.e., considering the instances of type T), and model (sBP) otherwise (i.e., considering the instances of type F). A time limit of one hour of CPU time was imposed also for CPLEX. Since in this phase of the research the focus was not on a direct comparison between CPLEX and our exact algorithm, we ran CPLEX with the standard set of parameters which has shown to be effective for most problems which do not have a particular structure. Indeed, although the proposed balanced path models exhibit a multicommodity flow structure, our preliminary computational results showed that the main reason of difficulty for CPLEX while solving the balanced path instances seems to be related to the objective function rather than to the flow constraints, and in particular this seems to be due to the absence of good lower bounds. The constraints relative to the objective function tend to destroy the network structure of the problem and, as we experimented in (Cappanera and Gallo 2004), in such cases CPLEX does not take advantage from the use of options such as the network simplex option.

We also solved the relaxation of models (mBP) and (sBP) where the cycle preventing constraints are removed, in an attempt to assess the computational difficulty encountered by

CPLEX in addressing those types of cycle preventing constraints. Clearly, so doing, the found solutions may include some cycles, and their cost thus provide a lower bound to the optimum path balancing. Indeed, CPLEX generally solved the relaxations in the same computational time required to solve the exact formulations (mBP) and (sBP). The computational results reported in the following sections thus refer to (mBP) (for the instances of type T) and to (sBP) (for the instances of type F).

Before entering the presentation of the obtained results, also another issue of the way the computational experience is presented deserves some considerations and, in our opinion, it is worthwhile to be discussed: after running the exact and the heuristic color-coding algorithms on the two groups of instances, we made an effort to try to understand which are the issues that characterize critical instances for our approach, and we attempted to get some insights on the way the instances may be classified into groups that reveal similar behavior. At first glance there are at least three issues that seem to influence the performance of the color-coding algorithms: the value of U which, as we stated before, is an estimate of the longest path cost in the network, the value of Δ reported in Tables 1 and 2 and, finally, the quality of the best solution found in the initialization phase via the solution of a minimum cost flow problem or with the shortest path based procedure. We recall the reader that parameter U determines an upper bound on the number of copies of each original node in the cost-expanded network, and thus it influences its dimension; so it is reasonable to assume that the more U is bigger the more the instance is critical for the color-coding framework. On the other hand, parameter Δ and the value of the feasible solution eventually found in the initial phase hopefully reduce the width of the range where the binary search is conducted; so, again, it seems to be straightforward that instances characterized by a small range are easier to solve than instances with bigger range. Nevertheless, by looking at the computational results, we observed that the issues that make an instance critical are sometimes difficult to ascertain. For these reasons we decided not to report aggregated results (a part from the summarizing aggregated results concerning ExactCC), but we preferred to discuss the performance of the color-coding algorithms on subsets of instances. So, in the following we will give some snapshots of the computational performances which in our opinion give some insights on the efficiency of the algorithms for the two classes of instances.

All the experiments were carried out on a bi-processor AMD Opteron 246 2 GHz, with 2 GB of RAM and under the operating system Linux, kernel 2.6.13-1. All algorithms were coded in C++ and compiled with GNU gcc 4.0.1.

Table 3: The critical SndLib instances

					Cplex		ExactCC		
ProbName	n	m	k	Δ Red	Obj	Time	Init	Obj	Time
janos-euF	37	114	5	3,64	**109**	906,08	**170**	170	3600,00
janos-euT	37	114	5	3,64	134	0,97	170	134	1050,55
janos-usF	26	84	4	4,11	133	11,27	133	133	1402,84
janos-usT	26	84	4	4,11	133	0,39	133	133	198,72
ta1F	24	55	4	4,00	13	2,70	39	13	42,63
ta1T	24	55	4	4,00	36	0,01	∞	36	56,59
janos-us-caF	39	122	5	1,33	**104**	3600,00	143	**20**	3600,00
janos-us-caT	39	122	5	1,33	99	1758,07	143	99	1119,15
sunF	27	102	4	2,00	40	3600,00	44	40	1847,34
sunT	27	102	4	2,00	40	616,31	44	40	237,49

5.2.1. The SndLib instances

According to the criteria described above, over the 58 instances of the SndLib test bed we got the following aggregated results: 48 times ExactCC determined the optimum solution very rapidly. Specifically, 48 times the balanced path set was computed within 1 second, with an average running time of 0,05 seconds. It is remarkable to observe that, in the initial phase, when the minimum cost flow or the shortest path based procedure is performed, the optimum solution has been found 24 times; nevertheless in some cases ExactCC spent some time in certifying the optimality of the obtained solution.

The aggregated results are encouraging, so as to believe that the three heuristics previously described might be able to reduce the computational time required on the most critical instances, while keeping low the loss in quality of the best solution found. We comment here on the subset of 10 peculiar Sndlib instances which proved to be most critical for our exact approach.

The critical Sndlib instances

The characteristics of these instances are reported in Table 3 giving the number n of nodes, the number m of arcs, the number k of paths to be balanced and the percent reduction of Δ with respect to U, namely Δ Red. With the aim of further investigating the performance of ExactCC we ran Cplex for these critical instances. In Table 3 for both the solvers Cplex and ExactCC we report the objective function value with the corresponding time (expressed in seconds). For ExactCC also the feasible solution value computed in the initialization phase is reported in column Init. Observe that, for the first 6 instances, CPLEX outperformed ExactCC: in particular 5 over 6 times ExactCC provided the optimum solution in much longer times; in one case, that is for the *janos-euF* instance, ExactCC provided a worse solution, with value equal to 170 with respect to the optimum solution value 109 computed by CPLEX (see bold entries in the first row of the table). Except for this instance the computational times required by CPLEX are limited (not over 12 sec.) while for ExactCC they can be quite high. Observe also the for the *ta1T* instance the search for a feasible solution in the initialization phase failed and nevertheless ExactCC was able to compute the optimum in less than one minute, which is however a huge time compared with the 0.01 seconds required by the commercial solver.

The last 4 instances seem to be quite difficult for both solvers; however for 3 instances ExactCC computed the optimum solution more rapidly than CPLEX and for one instance, namely *janos-us-caF*, it produced a solution of much better quality, i.e., a solution with value 20 instead of 104 (see bold entries in Table 3 - second group of instances). Observe that ExactCC was not able to certify the optimality of such a solution since it exceeded the time limit of one hour.

If we look at the characteristics of the 10 critical Sndlib instances and compare them to the ones of the other instances in the Sndlib data set (see Table 1), it appears that these instances are characterized by a large value of the estimate U jointly with a large value Δ. It comes thus into evidence that the size of the cost-expanded network, expressed in terms of estimate U of the longest path cost, and the difference between the longest and the shortest path cost in the network, given by Δ, indeed identify a critical issue for our exact approach in addressing the Sndlib data set. Observe also that for the first 6 instances, which are more critical for ExactCC, the Δ reduction, Δ Red, is around 4, whereas for the last 4 instances it is less than 2: so, the Δ reduction alone does not directly influence the efficiency of the exact color-coding algorithm.

For the first 6 instances, the most critical ones, an attempt was done to reduce the computational burden of ExactCC via the *5+5* heuristic. The results are reported in Table 4. For all these instances the heuristic, named Heur5+5, gave the same solution value of ExactCC so that in the table the comparison is done only in terms of time. The reduction of the time required by Heur5+5 with respect to ExactCC was meaningful at least for the first three most critical instances; in fact, the time required by Heur5+5 is around 10 minutes. Nevertheless the computational times of the *5+5* heuristic are still not competitive. In the

Table 4: Timing of the *5+5* heuristic

	Computational Times		
ProbName	Cplex	ExactCC	Heur5+5
janos-euF	906,08	3600,00	602,1
janos-euT	0,97	1050,55	698,24
janos-usF	11,27	1402,84	600,58
janos-usT	0,39	198,72	146,92
ta1False	2,70	42,63	42,63
ta1True	0,01	56,59	56,59
Average Time	153,57	1058,56	357,84

last row of the table also average timing information are reported: the ratio between the time required by ExactCC and the one required by Heur5+5 is almost 3, but on average the time required by Heur5+5 is still too large (it is two times bigger than the time required by the commercial solver).

So, for the 10 critical instances in this test bed we tested the *cost scaling* heuristic with $\varepsilon \in \{3/2, 2, 5/2, 3, 10\}$: the first 4 values of ε are close the one to the other with a step size of $1/2$ whereas for the last value we exaggerate the effect of the approximation with the aim to understand the extent up to which the loss in quality of the best solution found is still acceptable. The results are reported in Table 5, where computational times are approximated to the nearest integer when greater than 10 sec. For each instance and for each value of ε we report the time required by the cost scaling heuristic and the percentage relative error (column Gap) of the solution reported by the heuristic with respect to the best computed solution (determined either by ExactCC or by CPLEX). We report also the times required to determine the best computed solutions (column Best).

The results in Table 5 show that the cost scaling heuristic generally behaved very efficiently, producing in short time good quality solutions, which very often qualified as the optimum ones. More in detail, for the first three instances the cost scaling heuristic was not so effective in terms of time for any of the ε value in the range chosen. Anyway we would like to stress the fact that for the *janos-euF* instance with $\varepsilon \in \{3/2, 2, 5/2\}$ this heuristic was able to find an optimal solution of value 109, while ExactCC reported only a solution with a gap of 55.96% away from the optimum, as we can see from Table 3. For the other two instances in this group, namely *janos-euT* and *janos-usF*, observe that with $\varepsilon = 5/2$ the optimality of the solution is maintained and the computational times are reduced of a factor

Table 5: The cost scaling heuristic

	Best	$\varepsilon = 3/2$		$\varepsilon = 2$		$\varepsilon = 5/2$		$\varepsilon = 3$		$\varepsilon = 10$	
ProbName	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
janos-euF	906	0,00	3600	0,00	3600	0,00	3194	3,67	3280	26,61	327
janos-euT	0,97	0,00	254	2,99	161	0,00	79	2,99	73	2,99	9,86
janos-usF	11	0,00	299	0,00	162	0,00	130	0,00	100	3,01	1,73
janos-usT	0,39	0,00	19	0,00	9,09	0,00	5,79	0,00	4,00	0,00	0,20
ta1F	2,70	0,00	2,18	0,00	2,14	38,46	0,94	38,46	0,90	176,92	0,19
ta1T	0,01	0,00	3,82	0,00	1,60	0,00	0,91	0,00	0,75	0,00	0,02
janos-us-caF	3600	0,00	3600	0,00	2141	0,00	1665	15	1431	20,5	115
janos-us-caT	1758	0,00	333	0,00	216	0,00	183	0,00	118	5,05	6,47
sunF	3600	0,00	393	0,00	352	0,00	216	0,00	144	100,00	1,38
sunT	616	0,00	38	0,00	30	0,00	20	0,00	13	10,00	0,32

bigger than 10 with respect to ExactCC. Further observe that, for the first and the third instance, the cost scaling heuristic with $\epsilon = 10$ was able to compute solutions with a gap of 26.61% and of 3.01%, respectively, in about 1/3 and 1/10 of the time required to determine the optimum solution. Still better results were obtained for the 7 remaining instances. In the first 3 cases the cost scaling heuristic proved to be better than or equivalent to the best exact solver in determining the optimum solution. For *janos-us-caF*, either the heuristic computed the best solution, of value 20, in about half a time the time required by ExactCC (1665 seconds), or it determined a less good solution, with a gap of $20,5\%$, in 115 seconds. In the last 3 cases the optimum solutions were determined in a very short time setting $\epsilon = 3$.

In conclusion, concerning the tool number of colors, i.e., the ν-heuristic, it has to be emphasized that no significant improvement was obtained for the Sndlib test bed by choosing a number of colors $\nu < n$ (precisely, $\nu = 70\%n$ and $\nu = 80\%n$ were investigated). So, the related computational results are not reported here.

5.2.2. The Belotti instances

As for the SndLib instances, we begin by giving an account of the aggregated results concerning the exact color-coding algorithm. Over the 104 instances belonging to the Belotti data set, we got the following figure: 53 times ExactCC determined the optimum solution very rapidly. Specifically, 53 times the balanced path set was computed within 10 seconds, with an average running time of 1,26 seconds. Out of the 104 instances, the best solution

Table 6: Results for 7 instances of bhv and bvh type

				Cost scaling					
		ExactCC		$\varepsilon = 1/2$		$\varepsilon = 1$		$\varepsilon = 3/2$	
ProbName	CplexTime	Gap	Time	Gap	Time	Gap	Time	Gap	Time
bhv3cT	47,71	0,00	3600	0,00	195,50	0,00	35,40	0,00	11,94
bvh8cF	5,39	0,00	3600	0,00	62,70	0,00	8,33	0,00	4,75
bvh8cT	0,08	0,00	3600	0,00	58,98	0,00	7,60	0,00	4,36
bvh9cF	82,99	0,00	3600	0,00	27,03	0,00	6,88	0,00	2,31
bvh9cT	0,54	0,00	3600	0,00	42,67	4,82	8,24	0,00	2,77
bhvdcF	61,43	0,00	3600	0,00	83,60	0,00	22,63	8,89	5,62
bhvdcT	0,06	0,00	3600	0,00	143,97	0,00	31,00	0,00	8,67
AverageData	28,31	0	3600	0	87,78	0,69	17,15	1,27	5,77

was found 52 times in the initialization phase via the minimum cost flow or the shortest path based heuristic. The Belotti instances appear thus to be more critical than the SndLib ones for ExactCC, and a more comprehensive analysis is thus required.

As we discussed before, we report detailed computational results on subsets of instances which, according to our opinion, highlight the main criticisms or the benefits in using the heuristic color-coding algorithms.

Behavior on some bhv, bvh and net instances

We first examine the computational results on a subset of 17 instances, characterized by a large value of U, on which ExactCC always exceeded the time limit giving the optimal solution without however certifying its optimality. On these critical instances, CPLEX returned the optimum in computational times comprised between 0.06 sec. and 1884,04 sec (in 292.69 sec. on average). It is thus straightforward to observe that while these instances seem to present the same level of difficulty for our exact algorithm, their performance greatly varies for the commercial solver. The computational results relative to these instances are given respectively in Table 6, for a set of 7 instances of *bhv* and *bvh* type, and in Table 7 for a set of 10 instances of *net* type. The two tables show the same information: in particular they report the performances obtained with CPLEX, ExactCC and the cost scaling heuristic for $\varepsilon \in \{1/2, 1, 3/2\}$ in terms of time and, for ExactCC and for the cost scaling heuristic, in terms of percentage relative error with respect to the optimum value (columns Gap).

Observe that for 3 instances of *bhv* and *bvh* type, the cost scaling heuristic with $\varepsilon = 3/2$ (the largest value in the selected set) is not very effective in terms of computational time: in

Table 7: Results for 10 instances of net type

				Cost scaling					
		ExactCC		$\varepsilon = 1/2$		$\varepsilon = 1$		$\varepsilon = 3/2$	
ProbName	CplexTime	Gap	Time	Gap	Time	Gap	Time	Gap	Time
net_15_99F	8,23	0,00	3600	0,00	4,66	31,37	1,62	11,07	0,64
net_20_5F	886,98	0,00	3600	0,00	133,82	0,00	45,98	0,00	31,96
net_20_5T	4,04	0,00	3600	0,00	34,71	0,00	7,73	0,00	5,23
net_20_20F	1884,04	0,00	3600	0,00	26,53	26,47	14,30	26,47	3,56
net_20_20T	15,10	0,00	3600	0,00	10,17	4,85	3,49	76,70	0,96
net_20_60F	594,74	0,00	3600	0,00	78,59	0,00	25,03	0,00	18,61
net_20_60T	9,10	0,00	3600	0,00	26,35	0,00	5,88	0,00	3,05
net_20_99F	1103,86	0,00	3600	0,00	143,29	0,00	53,85	0,00	35,96
net_20_99T	7,90	0,00	3600	0,00	36,71	0,00	8,89	0,00	4,58
net_25_60T	263,55	0,00	3600	0,00	851,61	0,00	193,76	0,00	119,63
AverageData	477,75	0	3600	0	134,64				

fact, for these three instances (namely *bvh8cT*, *bvh9cT* and *bhvdcT*) the computational times are respectively 4.36, 2.77 and 8.67 seconds as opposed to Cplex which returns the optimum in 0.08, 0.54 and 0.06 seconds, respectively. Nevertheless the results obtained with the cost scaling version are encouraging when compared with the times obtained with ExactCC which for this subset of instances reaches the time limit. Besides, as reported in the last row of Table 6, the average results indicate that the cost scaling heuristic with $\varepsilon = 3/2$ represents a good tradeoff between the loss in the solution quality and the time required since, on average, it gives a solution which is only the 1.27% worse than the optimum in 5.77 seconds compared with the 28.31 sec. of Cplex to get the optimum.

Even better results are obtained by the cost scaling heuristic on the 10 instances of *net* type (see Table 7). In this case only for one instance, namely *net_20_5T*, the cost scaling heuristic with $\varepsilon = 3/2$ was not so efficient in terms of time by obtaining the optimum in 5.23 sec. opposed to the 4.04 sec. needed by the commercial solver. The increase in computational time is however very limited. We stress the fact that on these 10 instances the case with $\varepsilon = 1/2$ preserves the optimality of the solution and stops in an average time (see last row of Table 7) of 134.64 sec. as opposed to the 477.75 seconds needed on average to determine the optimum solution. Differently from what happens for the *bhv* and *bvh* type instances, on some of these *net* type instances the performance of the approach may degrade as ε increases, producing solutions of quite bad quality (relative error greater than 26% starting from $\varepsilon = 1$ for three instances). However, in all the other cases either the optimum solution or good quality solutions are computed very rapidly.

By summarizing the above illustrated results, and to give an idea of the overall behavior of the cost scaling heuristic on this subset of 17 critical instances, we state that the average time to compute the optimum solution is 292,69 sec. for CPLEX and 115,35 for the heuristic with $\varepsilon = 1/2$; in such a case the gap on the objective function value is always 0 while it may degrade on some instances as ε increases; in all the other cases, either the optimum solution or good quality solutions are computed in a very short time.

In concluding this discussion, it is noteworthy to pay attention to the fact that, on this subset, the heuristic Heur5+5 terminated in an average time of 762.84 sec. compared with one hour required by ExactCC, by reporting a loss of the optimality in 3 cases (deterioration limited by 8.89%) and one failure.

The big gap case

We now investigate on a further subset of instances for which the performances obtained by ExactCC was quite poor. In Table 8 we report on computational results relative to a subset of 10 instances for which ExactCC provided a solution of unsatisfactory quality, and which proved to be critical also for the commercial solver; we refer to this case as to the *big gap case*. For these 10 instances, which are characterized by an average value of U equal to 10452,11 and by a number of arcs m comprised between 118 and 142, CPLEX outperformed ExactCC both in terms of quality of the solution and computational time. In the following we try to give some insights on such a behavior.

Concerning the computational time observe that ExactCC always exceeded the time limit: in more details, for one instance (*net_25_20T*) ExactCC was not able to find a feasible solution, and for the other 9 instances it was only able to find a solution with a large percentage gap with respect to the best solution known. On average such a gap, computed over the 9 instances, is equal to 246.71 (see last row of Table 8). Note that for one instance the best solution value found by our approach is 579 with respect to the value 53 returned by CPLEX (the gap is equal to 992.45%). This group of instances seems however to be critical also for CPLEX which reports an average time of 2824.87 sec. and which for 7 out of 10 instances exceeds the time limit without certifying the optimality of the solution found.

From this analysis it seems to come into evidence that the size of the cost-expanded

Table 8: The big gap case

					Cplex		ExactCC		
ProbName	n	m	k	U	Obj	Time	Obj	Time	Gap
net_25_5T	25	118	4	7740	122	213,84	156	3600,00	27,87
bhv3cF	29	124	4	12342	64	3600,00	85	3600,00	32,81
net_25_99T	25	124	4	10548	62	9,99	94	3600,00	51,61
net_30_5F	30	142	4	11753	20	3600,00	32	3600,00	60,00
net_25_99F	25	124	4	10548	51	3600,00	94	3600,00	84,31
net_25_60F	25	124	4	9018	27	3600,00	88	3600,00	225,93
net_30_5T	30	142	4	11753	54	3600,00	246	3600,00	355,56
net_30_99T	30	146	4	12627	227	3600,00	1112	3600,00	389,87
net_25_5F	25	118	4	7740	53	3600,00	579	3600,00	992,45
net_25_20T	25	116	4	8316	141	1,18	∞	3600,00	∞
AvgData				10452,11		2824,87		3600,00	246,71

network, expressed in terms of estimate U of the longest path cost and number m of arcs, indeed identifies a critical issue for our exact algorithm in addressing the Belotti data set. A quite meaningful number of arcs seems also to influence the performance of CPLEX.

Motivated by this reasoning, we report in Table 9 the computational results obtained with the cost scaling heuristic, which has been designed just to handle large cost-expanded networks. In particular, the results in Table 9 refer to the setting $\varepsilon = 1/2$, which corresponds to the first value of ε for which the heuristic showed an efficient behavior. In 2 out of the 10 instances of this group, namely *net_30_5F* and *net_30_99T*, the cost scaling heuristic did not result either in an improvement of the solution quality or in a reduction of the computational time. The results relative to these 2 instances are not shown in Table 9. On the other 8 instances the performance of the cost scaling heuristic was very good: in fact, in 5 cases the heuristic was able to match the best solution value, in one case it was able to determine a solution with a small gap of 3.77%, and in 2 cases it succeeded in finding a solution of much better quality with respect to the ones returned by the commercial solver (see the results relative to *net_25_99F* and *net_30_5T*). From the last row of Table 9, it is worthwhile to observe that the time required by the heuristic is less than 1/3 of the time required by ExactCC, and it is less than the half of the time required by CPLEX.

The effect of random coloring

Now we give an idea of the effect of the random coloring on the performance of the color-

Table 9: The big gap case: performance of the cost scaling heuristic

	Cplex		ExactCC			$\varepsilon = 1/2$		
ProbName	Obj	Time	Gap	Obj	Time	Gap	Obj	Time
net_25_5T	122	213,84	27,87	156	3600	**0,00**	122	100,04
bhv3cF	64	3600,00	32,81	85	3600	**0,00**	64	176,02
net_25_99T	62	9,99	51,61	94	3600	**0,00**	62	361,91
net_25_99F	51	3600,00	84,31	94	3600	**-13,73**	44	2308,26
net_25_60F	27	3600,00	225,93	88	3600	**0,00**	27	2720,93
net_30_5T	54	3600,00	355,56	246	3600	**-50,00**	27	1324,74
net_25_5F	53	3600,00	992,45	579	3600	**3,77**	55	434,47
net_25_20T	141	1,18	∞	∞	3600	**0,00**	141	860,03
AverageTime		2278,13			3600			1035,80

Table 10: The effect of random coloring

		Cplex		ExactCC		RandomCol	
ProbName	Colors	Obj	Time	Obj	Time	Obj	Time
toronto_5F	70%	10	3600	10	80,87	10	17,37
toronto_25F	70%	10	1933	10	46,35	10	9,29
net_30_20F	80%	137	3600	86	3600	86	667,53

coding approach within the Belotti data set. In particular, in Table 10 the performance of ExactCC and of the ν-heuristic is given for a set of 3 instances which were quite difficult to solve for ExactCC. Also the performance of Cplex is reported. As far as the random coloring is concerned, for all the 3 instances we investigated a number of colorings α equal to 5, while the number of colors ν is 70% of the number of nodes n for the first two instances in the table and 80% of n for the last instance. It deserves attention to note that, for the most critical instance among these three ones, namely *net_30_20F*, ExactCC returned a solution of much better quality than the one given by Cplex (solution value of 86 instead of 137) even if both solvers reached the time limit. The ν-heuristic maintained the same good level of the solution quality while reducing the time of about 5 times.

On the first two instances of *toronto* type, the effect of the random coloring consisted in further reducing the computational time with respect to ExactCC (which in turn determined the optimum solution more rapidly than Cplex), while preserving the optimality of the computed solution.

Some further insights on the performance of the color-coding algorithms

The computational experience on the Belotti data set seems to suggest that big values of the estimate U of the longest path cost and of the number m of arcs, both characterize difficult instances, at least for the exact color-coding algorithm. It seems also to come into evidence that a big number of arcs alone is not so critical. To corroborate such a hypothesis we present the computational results for a subset of 12 instances of *latadl* and *net* type which are characterized by the biggest number of arcs among the Belotti instances. We tested both ExactCC and the cost scaling heuristic.

The behavior of ExactCC is illustrated in Table 11. For each instance, the dimension of the underlying network is given in terms of number n of nodes, number m of arcs and number k of paths to be balanced; then, an idea of the cost-expanded network size is reported in terms of U (observe that, within the Belotti data set, value Δ is generally of the same order of U). The last three columns show respectively the time required by CPLEX, the time required by ExactCC and the percentage relative error of the solutions reported by ExactCC with respect to CPLEX.

These 12 instances are difficult to solve for CPLEX, which exceeded the time limit for all the instances and reported worse solutions than the ones given by ExactCC (see negative values in column Gap) in 9 out 12 cases. In particular, for three of these instances (see bold entries in Table 11), CPLEX failed to find a feasible solution, whereas ExactCC terminated in at most 11.13 sec. by solving the problem to optimality.

There are four instances which seem to be difficult to solve for ExactCC, too, which reaches the time limit. For three of such instances ExactCC was still able to determine solutions of better quality than the ones found by the commercial solver, but it was not able to certify their optimality within the time limit considered. It is also remarkable to observe that on such instances the Heur5+5 heuristic obtained the same solution values as ExactCC in about 10 minutes.

Except for the four critical instances, the time required by the exact color-coding algorithm varies between a minimum of 3.06 and a maximum of 76.37 sec. The stated computational results thus reveal that, at least within the Belotti data set, instances with a value of U less than 2000 seem to be easy to solve for our exact algorithm (first group of instances, with computational time not over 12 sec.) even if they have a quite high number of arcs, whereas as soon as U increases the instances can be burdensome to solve. The cost scaling heuristic showed the same kind of behavior, although it performed more efficiently than ExactCC. For

Table 11: Identifying some criticism

ProbName	n	m	k	U	CplexTime	ExactCCTime	Gap
latadl_5F	39	172	5	1950	**3600**	**11,13**	$-\infty$
latadl_5T	39	172	5	1950	3600	4,07	-19,30
latadl_25F	39	172	5	1950	**3600**	**5,12**	$-\infty$
latadl_25T	39	172	5	1950	3600	3,42	0
latadl_99F	39	172	5	1950	3600	4,33	0
latadl_99T	39	172	5	1950	**3600**	**3,06**	$-\infty$
net_30_20F	30	142	4	13685	3600	3600,00	-37,23
net_30_20T	30	142	4	13685	3600	3600,00	-7,53
net_30_60F	30	146	4	12052	3600	76,37	-51,04
net_30_60T	30	146	4	12052	3600	48,68	-39,23
net_30_99F	30	146	4	12627	3600	3600,00	-37,56
net_30_99T	30	146	4	12627	3600	3600,00	389,87

example, setting $\varepsilon = 3/2$ the cost scaling heuristic determined the optimum solution, for the instances of *latadl* type, in about 1/3 of the time required by ExactCC. For the instances of *net* type (so, when U increases), the heuristic was able to improve the solution of ExactCC for the instance *net_30_20T* (value 54 as opposed to 86, in 1208,36 sec.), and it determined the optimum solutions for the instances *net_30_60F* and *net_30_60T* respectively in 0,57 and 0,38 sec. No improvement was obtained, on the contrary, for the remaining three instances of *net* type.

In addition to the subset of instances illustrated in Table 11, there are further 13 instances (with $U \in [162, 1950]$) for which ExactCC provided an optimum solution in an average time of 11 sec. On this additional set of instances, 6 over 13 times CPLEX reached the time limit of one hour while 6 over 13 times ExactCC provided an optimal solution in less than 1 sec. Such results seem thus to corroborate the thesis according which Belotti instances characterized by a small value of parameter U are easy to solve for the exact color-coding algorithm.

5.3. QoS properties of the generated balanced paths

As outlined by Property 4.1 in Section 4, the exact color-coding algorithm always generates a set of balanced paths which minimizes the cost of the longest path. That is, by restricting the attention to the set of the optimum paths, the one returned by the exact color-coding algorithm has the minimum longest path cost. Indeed, also the tested color-coding heuristics produced sets of paths with a limited longest path cost. For example, by considering the cost scaling heuristic, in the Sndlib data set there was an average deviation of 0,53 from the longest path cost returned by the exact color-coding algorithm (in 204 over 214 experiments the heuristic returned the same longest path cost as the one determined by the exact color-coding algorithm, and in one case it returned a lower one). As far as the Belotti data set is concerned, the cost scaling heuristic produced a larger average deviation, equal to 9,7, from the longest path cost returned by the exact color-coding algorithm. However, in 280 over 341 experiments the heuristic returned the same longest path cost as the one determined by the exact color-coding algorithm, and in 22 cases it returned a lower longest path cost.

Now, an interesting question is whether the computed balanced paths are "sufficiently" short from an absolute point of view, that is with respect to the minimum cost achievable by the sets of node-disjoint paths linking the given origins to the given destinations. This is particularly relevant in telecommunication applications, where minimizing the maximum hop cardinality and minimizing the longest path cost are very requested Quality of Service (QoS) requisites (LeBlanc, Chifflet and Mahey, 1999). For the first requisite see for instance Bley (2003) and Huygens, Mahjoub and Pesneau (2004); for the latter one, the interested reader is referred to (Itai, Perl and Shiloach, 1982) and (Li, McCormick and Simchi-Levi, 1990).

Hereafter it will be shown that the disjoint balanced paths computed by the color-coding algorithms are quite "short" on the average, both considering the maximum hop cardinality and the longest path cost. So, they well satisfy the above stated QoS constraints.

As far as the first QoS requisite is concerned, the average maximum hop cardinality of the computed balanced paths proved to be very low; in fact, it was 3 both for the Sndlib and the Belotti data sets (the average number of network nodes is 24 for the Sndlib data set, and 23 for the Belotti data set).

Consider now the latter QoS requisite, i.e., the longest path cost. Determining a set of disjoint paths characterized by the minimum longest path cost is NP-Hard also in the special case of acyclic networks (Li, McCormick and Simchi-Levi, 1990). So, a lower bound has been determined as follows. Given the network G, let min denote the shortest path cost in G from an origin node in $\{s_1, s_2, \ldots, s_k\}$ to a sink node in $\{t_1, t_2, \ldots, t_k\}$. Furthermore, let opt denote the minimum balancing of the disjoint paths in G from $\{s_1, s_2, \ldots, s_k\}$ to $\{t_1, t_2, \ldots, t_k\}$ for the instances of type F, and between the origin-destination pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ for the instances of type T. Then, it is easy to observe that for both types of instances:

$$LB_1 = min + opt$$

provides a lower bound to the minimum longest path cost.

Now, given G, let mcf denote the minimum cost of a flow in G from $\{s_1, s_2, \dots, s_k\}$ to $\{t_1, t_2, \dots, t_k\}$ (mcf is determined, for the instances of type F, by the procedure aimed at determining an initial feasible solution). Again, it is easy to prove that

$$LB_2 = \frac{mcf + opt}{k}$$

provides an alternative lower bound to the stated minimum longest path cost.

$$LB = \max\{LB_1, LB_2\}$$

thus provides a lower bound to the minimum longest path cost, both for the single and the multicommodity scenarios.

Let max denote the longest path cost in the solutions returned by the color-coding exact algorithm. Using the lower bound LB, we estimated the percentage deviation of max from the minimum longest path cost according to:

$$\frac{max - LB}{LB} \times 100\%.$$

Considering the instances of type F, the average percentage deviation proved to be 59% for the Sndlib data set, and 50% for the Belotti data set. However, this average percentage deviation is lower than 20% for 17 over the 29 Sndlib instances, and for 31 over the 52 Belotti instances. Furthermore, the gap is zero for 11 Sndlib instances and for 17 Belotti instances.

On the other hand, by looking at the instances of type T, the longest path cost of the solutions returned by the exact color-coding method showed an average percentage deviation of 41% from the minimum one in the Sndlib dataset, and an average percentage deviation of 48% for the Belotti group. Again, this average percentage deviation is lower than 20% for 19 over the 29 Sndlib instances, and for 33 over the 52 Belotti instances. Furthermore, the gap is zero for 10 Sndlib instances and for 18 Belotti instances.

Both for the Sndlib and the Belotti data sets, the paths returned by the exact color-coding algorithm are so quite short also from an absolute point of view, showing an average percentage deviation $\leq 20\%$ from the above introduced lower bound for the majority of the tested instances (36 over the 58 in the first data set, and 64 over 104 in the second data set). Moreover, for 56 over the overall 162 instances the minimum possible longest path cost was achieved.

6. Conclusions and future research

In this work we studied the balanced path problem in the general case where the underlying graph may have any structure. Firstly, a general algorithmic framework has been described to solve the problem, which is based on the color-coding ideas for computing simple paths. Several color-coding algorithms have been designed starting from the proposed framework, including both an exact approach as well as alternative color-coding heuristics. The algorithms have been tested focusing on a telecommunication application context. In fact, two large families of balanced path instances have been generated starting from two sets of benchmark telecommunication instances: the instances of the library SNDlib, available at http://sndlib.zib.de/, and the benchmark instances available at ftp://ftp.elet.polimi.it/users/Pietro.Belotti/mcf/data/. The obtained computational results are very interesting: in several cases the exact color-coding algorithm produced the optimal solution very rapidly; on the most critical instances, some of the proposed heuristics, especially the cost scaling heuristic, proved to be able to generate high quality solutions in a very short time.

The obtained results will be extended to other relevant application contexts. Specifically, we plan to study the balanced path problem in the transportation setting, by addressing balancing aspects, related to an even distribution of the workload, in crew scheduling applications.

Acknowledgments

We thank Dr. Giovanna De Pascale for her valuable computational assistance.

References

R.K. Ahuja. 1997. The balanced linear programming problem. *European Journal of Operational Research* **101** 29–38.

Alon, N., R. Yuster, U. Zwick. 1995. Color-coding. *Journal of the ACM* **42** 844–856.

A. Bley. 2003. On the complexity of vertex-disjoint length-restricted path problems. *Computational Complexity* **12** 131–149.

Burke, E.K., P. De Causmaecker, G. Vanden Berghe, H. Van Landeghem. 2004. The state of the art of nurse rostering. *Journal of Scheduling* **7** 441–499.

Camerini, P.M., F. Maffioli, S. Martello, P. Toth. 1986. Most and least uniform spanning trees. *Discrete Applied Mathematics* **15** 181–197.

Cappanera, P., G. Gallo. 2004. A multi-commodity flow approach to the crew rostering problem. *Operations Research* **52** 583–596.

Cappanera, P., M.G. Scutellà. 2005. Balanced paths in acyclic networks: tractable cases and related approaches. *Networks* **26** 230–245.

Carraresi, P., G. Gallo. 1984. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *European Journal of Operational Research* **16** 163–173.

Desrochers, M., F. Soumis. 1988. A generalized permanent labeling algorithm for the Shortest Path Problem with Time Windows. *Information Systems and Operational Research* **26** 191–212.

Gamache, M., F. Soumis, G. Marquis, J. Desrosiers. 1999. A column generation approach for large-scale aircrew rostering problems. *Operations Research* **47** 247–263.

Huygens, D., A.L. Mahjoub, P. Pesneau. 2004. Two edge-disjoint hop-constrained paths and polyhedra. *SIAM J. Discrete Math.* **18(2)** 287–312.

Itai, A., Y. Perl, Y. Shiloach. 1982. The complexity of finding maximum disjoint paths with length constraints. *Networks* **12** 277–286.

LeBlanc, L.J., J. Chifflet, P. Mahey. 1999. Packet routing in telecommunication networks with path and flow restrictions. *INFORMS Journal on Computing* **11(2)** 188–197.

Lee, S.S., M. Gerla. 2001. Fault tolerance and load balancing in QoS provisioning with multiple MPLS paths. *Lecture Notes in Computer Science* LNCS 2092 155–169.

Li, C-L., S.T. McCormick, D. Simchi-Levi. 1990. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics* **26** 105–115.

Martello, S., W.R. Pulleyblank, P. Toth, D. de Werra. 1984. Balanced optimization problems. *Operations Research Letters* **5** 275–278.

Miller, C.E., A.W. Tucker, R.A. Zemlin. 1960. Integer programming formulations and traveling salesman problems. *Journal of the ACM* **7** 326–329.

M.G. Scutellà. 1998. A strongly polynomial algorithm for the uniform balanced network flow problem. *Discrete Applied Mathematics* **81** 123–131.