An Exact Algorithm for Robust Network Design*

Christoph Buchheim[1], Frauke Liers[2], and Laura Sanità[3]

[1] Technische Universität Dortmund, Fakultät für Mathematik, Vogelpothsweg 87, 44227 Dortmund, Germany
[2] Universität zu Köln, Institut für Informatik, Pohligstraße 1, 50969 Köln, Germany
[3] Institute of Mathematics, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

Abstract. Modern life heavily relies on communication networks that operate efficiently. A crucial issue for the design of communication networks is robustness with respect to traffic fluctuations, since they often lead to congestion and traffic bottlenecks. In this paper, we address an NP-hard single commodity robust network design problem, where the traffic demands change over time. For k different times of the day, we are given for each node the amount of single-commodity flow it wants to send or to receive. The task is to determine the minimum-cost edge capacities such that the flow can be routed integrally through the net at all times.
We present an exact branch-and-cut algorithm, based on a decomposition into biconnected network components, a clever primal heuristic for generating feasible solutions from the linear-programming relaxation, and a general cutting-plane separation routine that is based on projection and lifting. By presenting extensive experimental results on realistic instances from the literature, we show that a suitable combination of these algorithmic components can solve most of these instances to optimality. Furthermore, cutting-plane separation considerably improves the algorithmic performance.

Key words: robust network design, branch-and-cut, target cuts, non-linear flow problems

1 Introduction

Communication networks play a fundamental role in every-day life. Due to the huge growth of telecommunications services in the last years, the development of efficient methods for an optimal design of such networks is nowadays a crucial research area.

In a standard network design problem, we are given a network represented by a graph with non-negative costs on the edges, and we aim at routing a set D of

* Financial support from the German Science Foundation (DFG) is acknowledged under contracts Bu 2313/1–2 and Li 1675/1–2.

demands through the network at minimum cost. However, since in practical settings the set of demands is often subject to uncertainty and may vary with time, more accurate models recently have been defined in the literature. In particular, there is a well-studied class of *robust* network design problems, which assumes to have as input a family $\mathcal{D}$ of possible sets of demands to be routed, instead of just one set. The aim is to install minimum cost capacities such that every set of demands $D \in \mathcal{D}$ can be suitably routed. Robust network design problems of this kind have received a lot of attention in the network design community, see e.g. [6, 4, 13, 11, 15] and the recent survey of Chekuri [9].

In this paper, we focus on a *single commodity* robust network design (RND) problem. As an example, suppose that some clients wish to download some program stored on several servers. For a client, it is not important which server he or she is downloading from, as long as the demand is satisfied. Still, at different times of the day (e.g. morning/afternoon/evening), the demands may change (e.g. different clients show up), and we would like to design a network that is able to route all flow in all different scenarios.

Formally, we are given an undirected graph $G = (V, E)$ with costs $c_{ij} \geq 0$ for every edge $\{i, j\} \in E$, and k sets $\{D_1, \ldots, D_k\}$ of demands. A set D_t, also called a traffic matrix, specifies for each node $u \in V$ a value $b_u^t \in \mathbb{Z}$ of flow that the node wants to send ($b_u^t < 0$) or to receive ($b_u^t > 0$); one may also have $b_u^t = 0$. The goal is to install integral min-cost capacities $u \in \mathbb{Z}^E$ such that each traffic matrix D_t can be (non-simultaneously) integrally routed on G without exceeding the capacity.

Note that, if we have only one traffic matrix (i.e. $k = 1$), then the problem is just a min-cost single commodity flow problem, the so-called transshipment problem, and it is therefore easily solvable in polynomial time (see e.g. [10]). In contrast, whenever we take into account more scenarios, the problem becomes NP-hard, already for $k = 3$ [25] (the complexity is open for $k = 2$).

To the best of our knowledge, no exact methods are available in the literature for this problem so far. In this work, we provide a branch-and-cut algorithm, based on the natural flow formulation strengthened by generating local cuts, revisited according to [7]. We test our algorithm on a wide set of realistic instances, and show that in this application local cuts significantly improve the computational time to find an optimal integral solution.

1.1 Related works

Robust network design problems with a family $\mathcal{D}$ of demand sets are widely studied in the literature.

A popular model is the one introduced by Ben-Ameur and Kerivin [6], where the family $\mathcal{D}$ is described by a polyhedron. In this setting, a well-known polyhedral set of traffic matrices is the so-called *hose model*, defined by Duffield et al. [11] and by Fingerhut et al. [15]. In fact, this model is at the basis of one of the most important robust network design problem, namely the Virtual Private Network Design problem [19, 18]. For robust network design problems with a

polyhedral set of traffic matrices, many exact algorithms (see e.g. [4, 13]) as well as approximation algorithms (see e.g. [12, 16, 20]) have been developed.

Whenever the set $\mathcal{D}$ of traffic matrices given in input is a finite list, as in our setting, the problem is a network synthesis problem with non-simultaneous flows. This problem has two main applications (see [22]): the first one, that we discussed in the previous section, is related to the design of a network with time-varying demands. Here the number k of different sets can be assumed reasonably small, but typically we have multiple sources/destinations. The second application is related to the design of *survivable networks*. Here the number k is large, since it is equal to the number of edges of the graph, but we have a single source and a single destination at the time.

The first application has been studied in more detail in the multi-commodity case (i.e. when each traffic matrix specifies a demand for every pair of nodes). This problem is NP-hard already for $k = 2$ [25]. In this setting, although k can be assumed to be small, a flow formulation would use different flow variables for every pair of nodes and every scenario, which make exact approaches based on flow formulations more difficult to solve in practice when comparing it to our single commodity setting. Some heuristics are given in [22]. Still, the problem can be approximated within a factor of $O(\log |V|)$ using metric embedding techniques [14, 25].

In survivable network design, we have a demand $r(i, j)$ for every edge $\{i, j\} \in E$ representing the flow that needs to be re-routed in case the edge $\{i, j\}$ fails, and the problem is to install capacity in order to non-simultaneously route each $r(i, j)$. In this setting, every flow is a single-commodity flow with exactly one source and one destination, but the number k is equal to the number of edges in the graph, therefore in a flow formulation we may again have order of $|V|^2$ different flow variables. The study of this problem was started already by Gomory and Hu [17], who provided combinatorial algorithms for finding an optimal fractional solution in unit-cost metric graphs. Later on, some people studied the polyhedral structure of the problem (see e.g. [24, 23]), and exact approaches for special classes of graphs (see e.g. [24, 26]).

Interestingly, there is a 2-approximation for the survivable network design problem due to Jain [21] that is based on an iterative rounding technique. Although our RND problem is also a single-commodity flow problem, we want to remark that, our setting has more sources and more destinations, which is different from survivable network design. This may make the problem harder to approximate. In fact, the 2-approximation algorithm given by Jain does not apply in our case (see [25] for more details), and it is an interesting open question to find a constant factor approximation for the single commodity RND problem.

2 Problem Formulation

We are concerned with the problem of assigning minimum cost edge capacities such that k different flows can be non-simultaneously integrally routed through the network. Clearly, once we compute k flows which realize the demands of the

k traffic matrices in input, the capacity which needs to be installed on an edge is just as large as the maximum amount of flow routed along it for all matrices.

For the matrix D_t, let the variable f_{ij}^t model the amount of flow that is routed along edge $\{i,j\}$ in the direction from i to j. Then our optimization problem, which from now on we simply call the RND problem, can be formulated as follows:

$$\begin{array}{l} \min \sum_{\{i,j\}\in E} c_{ij} \max(f_{ij}^1 + f_{ji}^1, \ldots, f_{ij}^k + f_{ji}^k) \\ \quad \sum_{j:\{j,i\}\in E} f_{ji}^t - \sum_{j:\{i,j\}\in E} f_{ij}^t = b_i^t \quad \text{for } i \in V,\ t = 1,\ldots,k \\ \qquad\qquad f_{ij}^t \in \mathbb{Z}_+ \quad \text{for } \{i,j\} \in E,\ t = 1,\ldots,k \end{array} \tag{1}$$

Considering the above set of constraints with linear cost functions, effective algorithms exist for determining optimum flows. However, in this formulation of the RND problem, the cost function is non-linear in the flow variables, which prevents their applicability. Trivially, this non-linear formulation can be linearized by introducing a capacity variable u_{ij} for each edge $\{i,j\} \in E$ that models the maximum amount of flow sent along the edge for all matrices:

$$\begin{array}{l} \min \sum_{\{i,j\}\in E} c_{ij} u_{ij} \\ \quad \sum_{j:\{j,i\}\in E} f_{ji}^t - \sum_{j:\{i,j\}\in E} f_{ij}^t = b_i^t \quad \text{for } i \in V,\ t = 1,\ldots,k \\ \qquad\qquad u_{ij} \geq f_{ij}^t + f_{ji}^t \quad \text{for } \{i,j\} \in E,\ t = 1,\ldots,k \\ \qquad\qquad f_{ij}^t \geq 0 \quad \text{for } \{i,j\} \in E,\ t = 1,\ldots,k \\ \qquad\qquad u_{ij} \in \mathbb{Z}_+ \quad \text{for } \{i,j\} \in E \end{array} \tag{2}$$

Note that in the above formulation we relaxed the integrality constraints for the flow variables. In fact, once an integral feasible capacity vector u is given, one can easily compute integral flows realizing our demands by solving k different flow problems, one for each traffic matrix D_t, with the given capacities u_{ij}. The existence of integral flows is guaranteed since we are dealing with single commodity flows (see e.g. [10]).

The linear relaxation of (2) is the LP at the basis of our branch-and-cut algorithm.

3 Preprocessing

We can preprocess a given RND instance by decomposing the network into biconnecting components. A biconnected component is a maximal connected subgraph such that the removal of any of its nodes does not destroy its connectedness. Any connected graph decomposes into its biconnected components, which are connected to each other by so-called cut vertices.

It is easy to see that the RND instance can be solved for each of the network's biconnected components independently as follows. There exists at least one of them, say $C = (V_C, E_C)$, that contains only one cut vertex $v \in V_C$. All flow into and out of C has to be routed through v. Therefore, we can decompose the RND

instance on G into an RND instance on C and an RND instance on a graph G' which is the union of all components different from C. Note that v is included in both C and G'. In the RND instance on C, the demand of the cut vertex is set to $b_v = \sum_{u \in V_C} b_u$. In the other instance, we set $b_v = \sum_{u \in V \setminus V_C} b_u$. The demands for all the other nodes are left unchanged.

Applying the same arguments to G' recursively and appropriately choosing the demands of the cut vertices, the RND problem on G can be reduced to RND problems on its biconnected components, which have a smaller size. The partial optimal solutions for different biconnected components can trivially be combined to an optimum solution for the whole network.

4 Primal Heuristics

Within a branch-and-cut approach, it is important to design so-called primal heuristics, that try guessing good feasible solutions to use as upper bounds on the optimum value. A standard approach finds feasible solutions by appropriately rounding the optimal solutions of the LP relaxations.

For the RND problem, suppose we have solved an LP relaxation at some node in the branch-and-bound tree, and let (f^*, u^*) be the optimum solution. We compute a feasible solution within three steps. First, we define $\overline{u} \in \mathbb{Z}^E$ to be the vector with entries $\overline{u}_e = \lceil u^*_e \rceil$, for all $e \in E$. Second, we determine k integral flows that satisfy the k different traffic matrices and respect the capacities given by $\overline{u}$. By construction such flows exist, and we compute them by solving a minimum-cost flow problem for each traffic matrix, with a randomly chosen linear objective function. Finally, the k flows are combined to an RND solution by determining for each edge the actual capacity u^{prim} necessary to route the k flows. Note that some entries of the vector u^{prim} could be strictly smaller than the corresponding entries of the vector $\overline{u}$.

In computing the k flows, we use the same cost function for all matrices, as the same edges should be preferred for each of the k flows, in order to keep the values of the capacity variables low. The cost function is chosen randomly in order to have the chance of generating different solutions in each iteration.

In the next lemma, we give an upper bound on the quality of a feasible solution $(f^{\text{prim}}, u^{\text{prim}})$ obtained by this procedure. More specifically, we relate it to the value of an optimum feasible solution contained in the subtree of the corresponding node of the branch-and-bound tree. If the node is not the root, we call such a solution local as it is optimum under the constraints given by the branching decisions.

Lemma 1. *The distance of a (local) optimum RND solution to the feasible solution $(u^{\text{prim}}, f^{\text{prim}})$ generated in the primal heuristic is at most $c^\top(\overline{u} - u^*)$.*

Proof. Let $(f^{\text{loc}}, u^{\text{loc}})$ be a (local) optimum solution that is feasible for RND. Clearly, $c^\top u^{\text{prim}} \leq c^\top \overline{u}$ and $c^\top u^{\text{loc}} \geq c^\top u^*$. This implies

$$c^\top u^{\text{prim}} - c^\top u^{\text{loc}} \leq c^\top \overline{u} - c^\top u^* \; .$$

5 Separation of Target Cuts

For designing an effective branch-and-cut algorithm, it is essential to separate strong cutting planes so that branching is rarely necessary. In [7], the separation of target cuts was introduced as a variant of the local cuts by Applegate et al. [5]. No predescribed structure is imposed on the generated cutting planes. Furthermore, their separation is a general procedure that can be applied in various contexts. For the separation, the problem is first projected into a low-dimensional space. Let $\overline{P}$ denote the convex hull of all projections of feasible solutions. Let x^* be the point to be separated and $\overline{x}^*$ its projection. The separation problem for x^* and the polytope P in question is solved heuristically by generating a facet separating $\overline{x}^*$ from $\overline{P}$, if it exists. Such a facet can be found by determining an optimal extremal solution of a linear optimization problem whose size is linear in the number of vertices of $\overline{P}$ and the extreme rays. Let $\overline{q} \in \overline{P}$ be arbitrarily chosen. The cut-generation LP is of the form

$$\begin{aligned} \max \quad & a^\top(\overline{x}^* - \overline{q}) \\ \text{s.t.} \quad & a^\top(\overline{x}_i - \overline{q}) \leq 1 \quad \text{for all vertices } \overline{x}_i \text{ of } \overline{P} \\ & a^\top(\overline{x}_i - \overline{q}) \leq 0 \quad \text{for all extreme rays } \overline{x}_i \text{ of } \overline{P} \\ & a \in \mathbb{R}^r \end{aligned} \tag{3}$$

If the optimum value of (3) is larger than 1, the inequality $a^\top(x - \overline{q}) \leq 1$ is violated by $\overline{x}^*$. Furthermore, it is proven in [7] that an optimum solution of (3) defines a facet of $\overline{P}$. In case (3) is unbounded, then $a^\top(x - \overline{q}) = 0$ is a valid equation for $\overline{P}$ and violated by $\overline{x}^*$, where a is an unbounded ray. Finally, the inequality is lifted to become valid (not necessarily facet-defining) for P. In [8], target cuts were successfully used for solving several constrained binary quadratic optimization problems.

5.1 Choice of the Projection

Choosing good projections is crucial for the success of the target-cut separation. In most optimization problems defined on a graph $G = (V, E)$, the polytope $\overline{P}$ is either determined through an orthogonal projection onto some subgraph of G, or through shrinking subsets of nodes or edges in G. The resulting graph is denoted by $\overline{G} = (\overline{V}, \overline{E})$.

For the RND problem, the polytope P in the original variable space is the convex hull of all feasible solutions (f, u) of problem (2). An orthogonal projection onto some subgraph that only contains a subset of nodes is not useful. Indeed, suppose $\overline{G}$ is obtained through an orthogonal projection such that for an edge $\{v_1, v_2\} \in E$ it is $v_1 \in \overline{V}$ but $v_2 \notin \overline{V}$. Then, $\overline{P}$ also contains vectors (f, u) for which f does not need to satisfy the flow-conservation constraints for v_1 because (positive or negative) excess flow at v_1 could be annihilated in G by routing flow along $\{v_1, v_2\}$. Therefore, most of the structure of the RND polytope is lost when using such a projection.

In contrast, we iteratively choose an edge $\{v_1, v_2\} \in E$ randomly and shrink it by identifying the nodes v_1 and v_2. Loops are deleted, multiple edges are replaced by one edge, the demand of the resulting supernode is set to $b_{v_1} + b_{v_2}$. The corresponding entries in the optimum solution x^* of the relaxation are summed up. We shrink until the number of (super-)nodes in the shrunk graph is equal to the value of a fixed parameter c that specifies the size of $\overline{G}$.

Let $\overline{P}$ be the convex hull of the vertices in the projected space obtained through shrinking edges as outlined above. Clearly, the vectors in $\overline{P}$ that are projections of feasible solutions of problem (2) need to satisfy the flow conservation constraints on $\overline{G}$. In fact, it is easy to see that $\overline{P}$ is again the convex hull of feasible solutions of problem (2), but defined on the graph $\overline{G}$.

Finally, a different parameter l specifies the number of traffic matrices that should be taken into account. In case this number is smaller than the original number k of matrices, we randomly choose a subset of them.

The target-cut separation routine now determines facets of $\overline{P}$ that are violated by $\overline{x}^*$, if they exist. These inequalities need to be lifted to become valid for P. We use standard lifting procedures. First, we argue that an inequality valid for a subset of scenarios remains valid if all scenarios are adressed. Indeed, as the capacity variables are not bounded from above in the RND model, their coefficients are necessarily nonpositive in any valid inequality. As the capacity values can only increase when enlarging the set of traffic matrices, an inequality that is valid for a subset remains valid when all matrices are considered. An inequality is then iteratively lifted to an inequality valid for P by simultaneously unshrinking the graph. The shrinking steps are undone one after the other, in reverse order of the shrinking procedure. Suppose some loop was deleted when shrinking an edge $\{v_1, v_2\}$. Furthermore, also suppose some multiple edges were replaced by a single edge e. We obtain the lifted inequality for the graph in which nodes v_1 and v_2 are unshrunk as follows. The coefficient corresponding to the loop edge is set to zero. Let $\overline{a}_e$ be the coefficient of the single edge that represents multiple edges. As the flow along e in $\overline{G}$ can now be split along multiple edges, $\overline{a}_e$ is set as coefficient for each of these. The coefficient of $\{v_1, v_2\}$ is set to zero. All other coefficients remain unchanged. It is easy to see that iteratively applying this lifting and unshrinking procedure yields an inequality valid for P.

5.2 Cut Generation

In our implementation, target cuts are generated by delayed column generation [7]. Starting from a small subset of feasible points in $\overline{P}$, the remaining points of $\overline{P}$ are generated only if necessary. More precisely, a candidate target cut is computed considering the initial set of points, then it is checked whether this inequality is violated by some point in $\overline{P}$ not generated yet. If so, the new point is added and a new candidate cut is computed. To check whether $\overline{P}$ contains a point violating the given cut, we need a so-called oracle that solves the RND problem on the shrunk graph $\overline{G}$. This is done by applying a branch-and-cut algorithm to the MIP model (2).

In our application, the delayed column generation approach is crucial. While for binary problems it might be a feasible approach to completely enumerate all integer points in $\overline{P}$ at once, at least in small dimensions, this is practically not possible any more in the presence of general integer variables, since the number of these point could become much larger. As described in [7], the delayed column generation procedure can be applied even if the set of initial points is low-dimensional. However, to avoid dealing with numerical problems and to speed up the cut generation process significantly, we always start from a full-dimensional polyhedron $\overline{P}_0$, which is computed as sketched in the following.

First we compute any feasible solution (f, u) for the RND problem on the shrunk graph $\overline{G}$ and set $\overline{P}_0 = \{(f, u)\}$. Such a solution (f, u) can be computed efficiently as a composition of arbitrary feasible solutions for the single matrices, computing appropriate values for the variables u_{ij} in the end. Next, we determine any cycle basis of $\overline{G}$. For each cycle C in the basis and for each of the l matrices considered (recall that we may select a subset of the k matrices), we add the incidence vector of C to the entries of f corresponding to the chosen matrix and adjust the u-entries. The result is a new feasible vector in $\overline{P}$, which we add to $\overline{P}_0$. All vectors added in this way are affinely independent. If $\overline{G}$ has $\bar{n}$ nodes and $\bar{m}$ *directed* edges (counted in both directions (i, j) and (j, i)), the cycle basis contains $\bar{m} - \bar{n} + 1$ elements, so the current polytope $\overline{P}_0$ has dimension $l(\bar{m} - \bar{n} + 1)$.

Additionally, we add an unbounded direction to $\overline{P}_0$ for each variable u_{ij}, since increasing u_{ij} preserves feasibility. Equivalently, in the cut generation LP (3) we may enforce that the coefficient of u_{ij} is non-positive. The dimension of $\overline{P}_0$ increases to $l(\bar{m}-\bar{n}+1)+\frac{1}{2}\bar{m}$. Finally, for each vertex in $\overline{G}$, we have a valid flow conservation constraint (and all but one of these equations are independent). If a is the coefficient vector of any such constraint, we add the unbounded directions a and $-a$ to $\overline{P}_0$. Considering (3), this implies that all generated target cuts will be orthogonal to all flow conservation constraints. The final dimension of $\overline{P}_0$ is $l(\bar{m} - \bar{n} + 1) + \frac{1}{2}\bar{m} + l(\bar{n} - 1) = (l + \frac{1}{2})\bar{m}$, which means $\overline{P}_0$ is full-dimensional.

If a target cut is found in the end, i.e., if $\overline{x}^* \notin \overline{P}$, we try to compute further target cuts by a reoptimization approach: in (3), we choose the first non-zero coefficient in the last generated cut and fix it to zero. Then we reoptimize (3), using delyed column generation again if necessary, and continue fixing coefficients until no violated target cut is found.

6 Computational Results on Realistic Instances

We implemented an exact branch-and-cut algorithm based on the ideas outlined in the previous sections, using the optimization tool SCIL in combination with ABACUS 3.0 [1], LEDA 6.1[3], and CPLEX 12.1 [2]. The executable is run on 2.3 GHz machines with a limit of four hours CPU time for each job. Furthermore, as the program is a 32-bit executable, a maximum of 4 GB of memory can be addressed. Parameters controlling the target-cut separation are the chunk size c and the number of traffic matrices l used in the target-cut separation.

In particular, for $c = 0$, no separation is performed. We evaluate our method on 1120 realistic network topologies from the literature [4]. For each of these instances, $k = 2, 3, 4, 5$ random traffic scenarios are added. Furthermore, for each network topology and each choice of k, we randomly choose the percentage p of terminals, i.e., nodes with non-zero demand, as $p = 25, 50, 75, 100\%$. For very small instances, we did not use $p = 25\%$. Altogether, these are 1120 different instances. In Table 6, we report the distribution of the instance sizes. As the instances from [4] strongly vary in size, they are grouped with respect to the number of nodes in the network in bins of size 150. Average node and edge numbers of the instances in the respective groups are also given.

bin	$\|V\|$avg	$\|E\|$avg	# instances
$0 \leq \|V\| \leq 149$	34.31	55.48	612
$150 \leq \|V\| \leq 299$	201.31	383.57	268
$300 \leq \|V\| \leq 449$	352.00	578.80	240

Table 1. Distribution of sizes for the realistic instances, grouped in bins by the number of nodes $|V|$ in the network.

Within the time and memory limits, 94% of the instances could be solved to optimality, even without separation. Using target cuts separation, we can further increase the number of instances that can be solved to optimality. The fact that almost all of the instance set can be solved shows the effectiveness of our approach. In order to evaluate the computational results in more detail, we first assess the quality of the primal heuristic. For the instances that could be solved to optimality without separation, we report the distance of the optimum solution to the feasible RND solution generated from the first LP relaxation. More specifically, we determine the relative gap g in percent, i.e.,

$$g = 100 \cdot \frac{x^{\text{prim}} - x^{\text{opt}}}{x^{\text{opt}}} \ .$$

In 24% of the cases, $g < 0.1\%$, in 45% of the cases, $g < 1\%$, and in 55% of the cases, $g > 10\%$. In the worst case, the gap is not larger than 61%.

In Table 2, we report results for solving the instances to optimality. Results are presented separately for each number l of scenarios, grouped with respect to the number of nodes in the network. As the running time usually increases for chunk sizes larger than 4, we restrict ourselves to smaller chunks and therefore use the parameter choices $(c, l) = (0, 0), (3, 2), (3, 3), (4, 2), (4, 3)$, where the case $(0, 0)$ means that no separation takes place. Numbers are only shown for instances in which the number k of scenarios is at least as large as l. For each parameter choice, we report in the first column the number of instances that could be solved to optimality for a specific choice of separation parameters, followed by the number of instances that could not be solved due to time or memory constraints, respectively. Typically, for an instance that could not be solved due

to memory constraints, a number of subproblems in the order of 10^5 was generated. The following columns show the average number of subproblems in the branch-and-bound tree and the average cpu time of the instances that could be solved to optimality.

Interestingly, many instances in Table 2 can be solved within a few minutes only. On average, instances with only two scenarios are computationally easier than those with a larger number of scenarios, as can be expected. Furthermore, on average the difficulty often increases with increasing network sizes.

Clearly, target-cut separation considerably improves the performance of the algorithm. Whereas several instances cannot be solved to optimality without separation, the number of unsolved instances is never worse and often better if target-cut separation is used. Furthermore, the instances that are solvable without separation can be solved considerably faster and within a smaller number of subproblems when target cuts are separated.

We conclude that our approach can solve to optimality most of the realistic instances that we have at hand. Whereas instances defined on small networks should probably be solved without separation, in many cases target cut separation leads to faster solution of instances or even makes it possible to solve otherwise unsolvable instances.

References

[1] ABACUS – A Branch-And-CUt System. www.informatik.uni-koeln.de/abacus.

[2] ILOG CPLEX 12.1, 2009. www.ilog.com/products/cplex.

[3] LEDA – Library of Efficient Data Types and Algorithms. www.algorithmic-solutions.com.

[4] A. Altin, E. Amaldi, P. Belotti, and M.C. Pinar. Provisioning virtual private networks under traffic uncertainty. *Networks*, 49(1):100–115, 2007.

[5] A. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, volume 2241 of *Lecture Notes in Computer Science*, pages 261–304. Springer Verlag, 2001.

[6] W. Ben-Ameur and H. Kerivin. Routing of uncertain demands. *Optimization and Engineering*, 3:283–313, 2005.

[7] C. Buchheim, F. Liers, and M. Oswald. Local cuts revisited. *Operations Research Letters*, 36(4):430–433, 2008.

[8] C. Buchheim, F. Liers, and M. Oswald. Speeding up IP-based algorithms for constrained quadratic 0–1 optimization. *Mathematical Programming (Series B)*, 124(1-2):513–535, 2010.

[9] C. Chekuri. Routing and network design with robustness to changing or uncertain traffic demands. *SIGACT News*, 38(3):106–128, 2007.

[10] V. Chvátal. *Linear programming*. Series of books in the mathematical sciences. W.H. Freeman, 1983.

$\|V\|$	(0,0)					(3,2)					(3,3)					(4,2)					(4,3)				
	slvd	t	m	subs	CPU	slvd	t	m	subs	CPU	slvd	t	m	subs	CPU	slvd	t	m	subs	CPU	slvd	t	m	subs	CPU
$0 \leq \|V\| \leq 149$	152	0	1	373.09	1.40	152	0	1	116.78	1.20						153	0	0	62.34	3.46					
$150 \leq \|V\| \leq 299$	63	1	3	4416.32	112.01	66	0	1	1272.03	133.91						65	0	2	932.94	331.68					
$300 \leq \|V\| \leq 449$	56	0	4	1093.98	1.64	60	0	0	3301.40	85.96						59	1	0	1370.51	228.32					
$0 \leq \|V\| \leq 149$	149	0	4	627.54	3.66	150	0	3	323.63	3.28	150	0	3	506.13	12.85	152	0	1	135.61	9.95	151	0	2	146.58	49.46
$150 \leq \|V\| \leq 299$	59	0	9	7764.75	57.86	61	3	4	237.79	8.42	61	3	4	458.61	67.93	61	3	4	371.62	120.32	59	5	4	409.24	511.14
$300 \leq \|V\| \leq 449$	57	0	3	8225.11	139.17	60	0	0	3780.33	108.29	58	2	0	3344.52	262.48	59	1	0	1928.53	336.32	56	3	1	938.36	522.55
$0 \leq \|V\| \leq 149$	146	0	7	844.09	7.53	147	0	6	409.96	7.29	148	0	5	1554.40	50.00	150	0	3	383.42	26.49	150	1	2	421.49	123.11
$150 \leq \|V\| \leq 299$	61	0	6	4403.41	23.66	62	0	5	1685.76	59.78	62	1	4	1277.21	166.56	60	3	4	1981.53	657.47	58	5	4	426.67	539.23
$300 \leq \|V\| \leq 449$	57	0	3	1723.42	3.65	58	0	2	579.90	17.66	58	0	2	630.38	34.01	59	1	0	1319.71	220.21	58	2	0	719.55	380.44
$0 \leq \|V\| \leq 149$	150	0	3	994.91	10.17	150	0	3	441.43	7.39	150	0	3	475.33	15.94	152	0	1	256.52	16.80	150	0	3	250.77	56.18
$150 \leq \|V\| \leq 299$	57	0	10	8762.09	99.26	61	2	4	4940.90	172.72	57	6	4	4005.89	409.68	58	5	4	2309.07	551.97	55	8	4	584.24	661.29
$300 \leq \|V\| \leq 449$	50	0	10	7774.72	69.81	55	0	5	6373.80	180.31	55	2	3	6450.38	421.98	54	5	1	3126.33	384.23	51	9	0	1148.20	734.27

Table 2. Experimental results for the realistic instances from the literature. From top to bottom, the number of scenarios increases from $k = 2$ to $k = 5$. For each parameter choice in the separation, we first report the number of instances solved, followed by the number of instances that were not solved due to the time limit (t) or due to memory constraints (m), respectively. The following columns show the average number of subproblems and the average cpu time for solving the remaining instances.

[11] N.G. Duffield, P. Goyal, A.G. Greenberg, P.P. Mishra, K.K. Ramakrishnan, and J.E. van der Merwe. A flexible model for resource management in virtual private networks. *Proceedings of SIGCOMM*, 29:95–108, 1999.

[12] F. Eisenbrand, F. Grandoni, G. Oriolo, and L. Sanità. New approaches for virtual private network design. *SIAM Journal on Computing*, pages 706–721, 2007.

[13] T. Erlebach and M. Rüegg. Optimal bandwidth reservation in hose-model VPNs with multi-path routing. *Proceedings of INFOCOM*, 4:2275–2282, 2004.

[14] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.

[15] J.A. Fingerhut, S. Suri, and J.S. Turner. Designing least-cost nonblocking broadband networks. *Journal of Algorithms*, 24(2):287–309, 1997.

[16] S. Fiorini, G. Oriolo, L. Sanità, and D.O. Theis. The VPN problem with concave costs. *SIAM Journal on Discrete Mathematics*, pages 1080–1090, 2010.

[17] R.E Gomory and T.C. Hu. Multi-terminal network flow. *SIAM Journal on Applied Mathematics*, 9:551–570, 1961.

[18] N. Goyal, N. Olver, and B. Shepherd. The VPN conjecture is true. *Proceedings of STOC*, pages 443–450, 2008.

[19] A. Gupta, J. Kleinberg, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. *Proceedings of STOC*, pages 389–398, 2001.

[20] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. *Proceedings of STOC*, pages 365–372, 2003.

[21] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1998.

[22] M. Labbe, R. Séguin, P. Soriano, and C. Wynants. Network synthesis with non-simultaneous multicommodity flow requirements: Bounds and heuristics, 1999.

[23] T.L. Magnanti and S. Raghavan. Strong formulations for network design problems with connectivity requirements. *Networks*, 45:61–79, 2005.

[24] T.L. Magnanti and Y. Wang. Polyhedral properties of the network restoration problem with the convex hull of a special case. Technical report, Operations Research Center, MIT, 1997.

[25] L. Sanità. *Robust Network Design*. Ph.D. Thesis. Università Sapienza di Roma, 2009.

[26] S. Sridhar and R. Chandrasekaran. Integer solution to synthesis of communication networks. *Mathematics of Operations Research*, 3:581–585, 1992.